Universiteit Leiden

ICT in Business and the Public Sector

Towards Microservices architecture: A case study to improve decision-making

Name: Seif Mohamed Farag
Student-no: 2142848

Date: 28/06/2021

1st supervisor: dr. W. Heijstek
2nd supervisor: dr. J.B. Kruiswijk

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Maintaining and scaling a system is a never-ending process for organizations of all industries. The introduction of cloud services and containers has facilitated the development of more scalable architecture solutions. Microservices, as one solution, is thought to help organizations maintain their systems. This thesis is focused on the study of the application of microservices in the financial sector, and the decision-making process. Since multiple companies have been trying to implement this technology, the literature provided some cases where some were in favor while the rest were not. Therefore, a qualitative research approach was taken to develop this thesis study with the help of an internship. A case study is presented to summarize the organization's effort in understanding its system and preparation for microservices implementation. The results have shown that the study period has taken longer than expected. However, a method has been developed and resulted in a blueprint showcasing the different aspects of the organization. Moreover, this process discovered several alternatives to existing processes. The organization is then left to decide whether to choose the new architecture or focus on the discovered alternatives.

Table of Contents

1. Introduction

With the continuous advancement in technology, businesses are expected to keep up with these changes which can be a challenge. Corporations, among other types of businesses, do their utmost to be up to date with the latest technology to excel in their day-to-day operations. These challenges do not only challenge their operational or functional level but also their infrastructure and core components. Further, competition is a circadian factor in every operational business, and with new technology being discovered regularly, change is then required, demandingly.

The financial sector is one of the most continuously changing industries as they are affected by the constant need to be reliable and secure. The rapid transformation that people witness within their financial services, especially the banks, from the paper-based type of agreements to the one-click transaction is with no doubt concerning. The banking sector has to keep their eyes open for the market and the changes around them, adapt, and provide their clients what is needed and more. (Stelligent)

Some changes take a day or two, some take a few months, and some take a couple of years. With the concerning change when it comes to this project, the priority and the time to be given is the longest (Wix Engineering, 2015). The topic of interest is enterprise architecture, the transformation of monolithic, more typical software as a service architecture to microservices. The architecture that is ruling the corporate world is heading towards being more distributed, easier to maintain, edit, and fix. This is where the microservices architecture becomes useful (Kratzke, 2018).

As monolithic architecture has been there for a while, maintaining a small portion of it is a part of a whole. In other words, an enterprise wants to ameliorate one of its processes or fix a problem in one of the departments; the whole system must be considered since it is one big system. It is hard to maintain and needs much attention, as it causes downtime of the whole system. (Dragoni, et al., 2017) Therefore, the need for change exists.

Microservices are originally brought up from the monolithic Service-oriented architecture (SOA) that companies are working with. However, it is a less complicated architecture that would help organizations maintain their services with less downtime and more scalability. This happens by breaking the services down and therefore maintaining each service on its own. The idea is to give each business service a block or a microservice that can be handled individually to ensure less challenging, time-consuming maintenance. (Bucchiarone, Dragoni, Dustdar, Larsen, & Mazzara, 2018)

Multiple companies from different industries have started to initiate the transformation, including Amazon and Netflix, among many other companies. (Yadav, 2018) These companies have shown success and good results, and therefore, it is worth investigating this transformation in the banking sector. Nevertheless, new technology can work for a specific sector or industry better than another. Meaning that the software architecture in question can be more beneficial to one case than another.

2. Objectives

Microservices are still being tested in the corporate world, and since companies are not always in favor of change, they do not tend to take a fast decision of switching to this architecture. In this study, microservices and monolithic architectures are described and discussed thoroughly. A comparison between both architectures, how to maintain them, the challenges and the benefits that result from both. The study is expected to conclude a guideline in deciding whether a transformation is beneficial or not. By giving methods and steps to understanding the critical transformation points, the organization chooses the best for their end goals.

Like any other software architecture, microservices cannot be the only solution to every problem since it comes with its challenges and difficulties, which should be taken into consideration. This thesis project aims at developing a way to determine the return on the investment put into the transformation. The efforts and the challenges faced by the teams facing the change of the infrastructure are some of the general dilemmas faced in such situations. What needs to be figured out is whether the difficulties currently faced by a company and its customers are worth a complete change. Furthermore, will that change create a better environment for both or more challenges?

The decision here comes after deciding which challenges the beneficiary is comfortable facing. When it comes to monolithic architecture, the challenge of having a significant software application with one big code that faces multiple single points of failure is a reason why the change is considered. With the high complexity of monolithic software, a microservices architecture could be seen as a potential solution by scaling the software down into small, singly managed services.

The thesis will test the findings on a real-time case in the banking sector that has already existed for a long time. The findings will provide insights on trial and error to prove the concept of the transformation from the existing monolithic application to microservices.

The results will be in the form of suggestions on how decision making of the software should evolve in the future. The analysis provided on the literature and the case study should provide a constructed plan on assessing the situation at hand and understanding the difficulty to be faced when deciding about the transformation.

3. Research Question

Defining a way to identify potential in implementing microservices architecture with respect to showcasing an understanding of the system and the ability of the team to take the initiative.

The banking sector alongside other sectors considers the implementation of microservices into their architecture. This thesis project is aiming at studying the technology and the business environment and provide a way to facilitate decision-making in regard to the implementation.

This research will study the technology and its different aspects in order to identify critical needs in the process of pre-implementing the microservice architecture within an organization. The research aims at answering the question:

"How can an organization decide on the potential implementation of microservices architecture based on their understanding of their system and the abilities of the development team."

To facilitate the process of answering the main question, a set of sub-questions are then prepared:

1- *What are the organizational conditions needed for an architecture change?*
2- *how to analyze the current situation to prepare for microservices migration?*
3- *How to identify opportunities for microservices migration?*

The thesis will answer this question by providing academic research and case studies in the literature review, followed by a case study derived from an organization in the banking sector through which a process will be followed, studied, and documented. The process will be then verified by the research and the results and generalized for the documentation of this thesis project.

Figure (1) is a visual representation of the planned process for this thesis study to reach an answer to the questions listed above. The process starts from the basic understanding of the domain in the banking sector and microservices technology. it also extends to understand the theory and the cases around the subject in order to reach a conclusion to the research question.

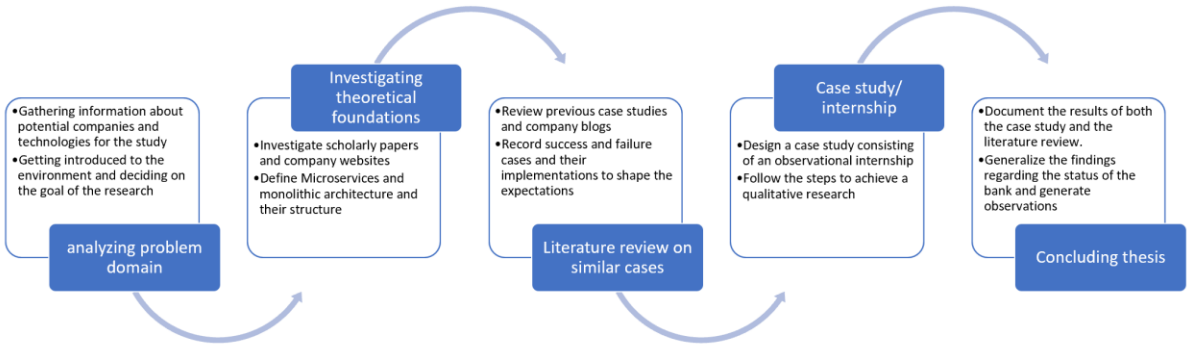


Figure (1) representing the main steps to answer the research question

4. Related Work

4.1. Microservices

Microservices is a cloud-native architecture that can achieve a software system through a set of small services that are independently deployed and singly run. Each service in that set represents a business capability on its own (Balalaie, Heydarnoori, Jamshidi, Tamburri, & Lynn, 2018). Microservices is not considered a new architecture style, especially when compared with Service Oriented Architecture (SOA). Following new trends and technologies, microservices are to be considered a direct evolution of SOA that has been showing success being introduced in different types of applications. (Nehme, Mahbub, Jesus, & Abdallah, 2019)

Moreover, Nehme et Al., (2019) mentioned in their paper that microservices allow the so-called Conway's law; this law entails that the processes and the workflow of the enterprise are followed closely within this architecture. They further explain the complexity of the application under this architecture which is narrowed into the number of small independent components that communicate together. Moreover, Al-Debagy and Martinek (2018) mentioned that communication within the architecture happens between the independent services through lightweight mechanisms such as HTTP resource and API, allowing each service to have its process.

Microservices introduce many features to the application that include cohesiveness; for each microservice, functionalities solely related to the business function's concern are implemented. Services are loosely coupled, allowing services to run without knowing much about the rest of the same application's different services. Each service is then autonomous, and an event broker is used to publish and store state-changing events achieving the level of communication needed to function, figure (2) illustrates an example of microservices architecture. (Al Debagy & Martinek (2018), Nehme, et Al., (2019), Barashkov, (2018))

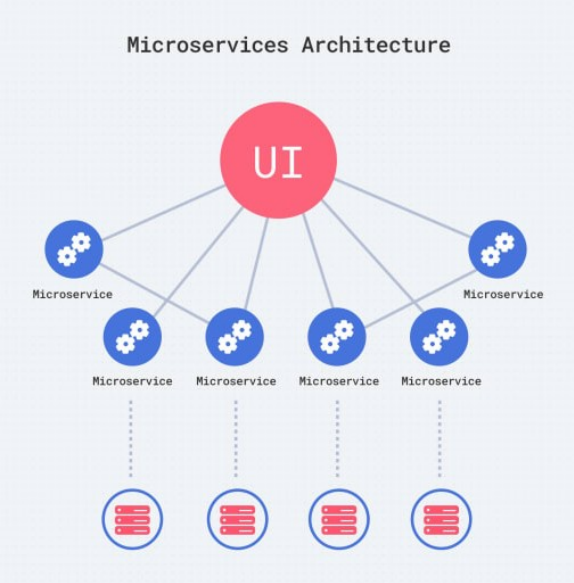


Figure (2) – Example of microservices architecture

Furthermore, Microservices are very scalable when it comes to the usage of the environment. The scalability here is then applied to the service level, not the whole system. Aligning architecture with organizational structure is then considered one of the features of microservices and its technological heterogeneity. Balalaie et al. (2018) mentioned how technology heterogeneity helps avoid technology lock-in and better comprehensibility of the codes. Failure of one service within this architecture does not mean the whole system will stop; this emphasizes the difference between microservices and currently employed technologies. (Al Debagy & Martinek (2018), Nehme, et Al., (2019))

Consequently, microservices allow avoiding the single point of failure or at least reduce its possibility. With high scalability and communication, services within the architecture avoid long deployment periods and reliability of different components within the system that reduces the time to fix issues and bugs. The independence of each service allows for minimal impact occurring due to to the failure of one service on the rest.

However, Nehme et Al. (2019) mentioned that the lowest spec server like the ones used in public cloud providers is ordinarily costly in covering for the resources of microservices. They also mentioned that setting up a virtual machine becomes a tricky matter when the number of dependencies increases. All that being mentioned, security seems to play a significant role in this architecture, since communication within the services is crucial but somewhat different.

Nehme et Al. (2019) discussed security in-depth in their paper, discussing how microservices are required to cooperate and communicate through and end-to-end coordination. Two mechanisms are introduced: orchestration, which represents the conductor, sending requests and organizing the workflow, the choreography, and where events and triggers allow services to reach accordingly. The model shown in the paper illustrated a gateway that handles provided requests externally, returning tailored responses based on the client type and access management by communication with an authorization server.

They further discussed different layers where security had to be considered and implement in the stages of deployment. The microservices' components, the application architecture, the operating system, and the network, and the external interfaces are used for interdomain communication. Securing microservices, therefore, depends on the implementation and usage of the application and its environment.

Within the process of implementing microservices, it is crucial to understand the need and the difference between the already employed architecture. Monolithic architecture, the most used architecture, is an extensive application that holds the code of different services together as one. The application maintains a high level of complexity, making it hard to maintain and continuously deploy.

Monolithic architecture has been used for a long time in enterprises that have witnessed long codes affecting the deployment period, making a notable effect on the whole application when a small

change is needed. The monolithic application's complexity is considered high, resulting in the need for an extended period of delivery since regular and fast cycles are therefore impractical. As mentioned earlier, when microservices are run independently, a single point of failure is less prone to happen, contrary to one powerful software or application.

Building a microservice from a starting point can be pricey and time-consuming as Fritzsch et Al. (2019) discussed in their paper about the transformation from monolithic to microservices. They discussed the refactoring process and its approaches trying to find the most suitable way. The process of migration was explained by Balalaie et al. as a process of trial and error. They preferred one approach called situational method engineering, which replaces the concept of one-size-fits-all.

Fritzsch et Al. (2019) further mentioned that monolithic architecture could become fossilized, meaning that the accumulated technical debt can result in obscure structures that make the product unmaintainable with a useful resource. The process of changing the initial design could require immerse effort. Duplicating instances of the whole application is considered as an option while maintaining high cohesion inward and loose decoupling outwards.

4.2. Monolithic Architecture

To understand the process of transformation from one architecture to another, both architectures should be explained thoroughly. In this section, an explanation of the more traditionally known and used architecture is explained, highlighting the reasons why it exists and how it comes into use.

Allowing the development of presentation, application, business logic, and data access object, the monolithic architecture helps to gather these components and gather them with enterprise or web archives (EAR and WAR). Sometimes even stores separately in a single directory such as NodeJs or Rails. Figure (3) represents an example of a layer monolithic application design used before Service Oriented Architecture (SOA). Applications were developed in this architecture in different companies such as Amazon and eBay, which later moved to Microservices. (Sharma, RV, & Gonzalez, 2017)

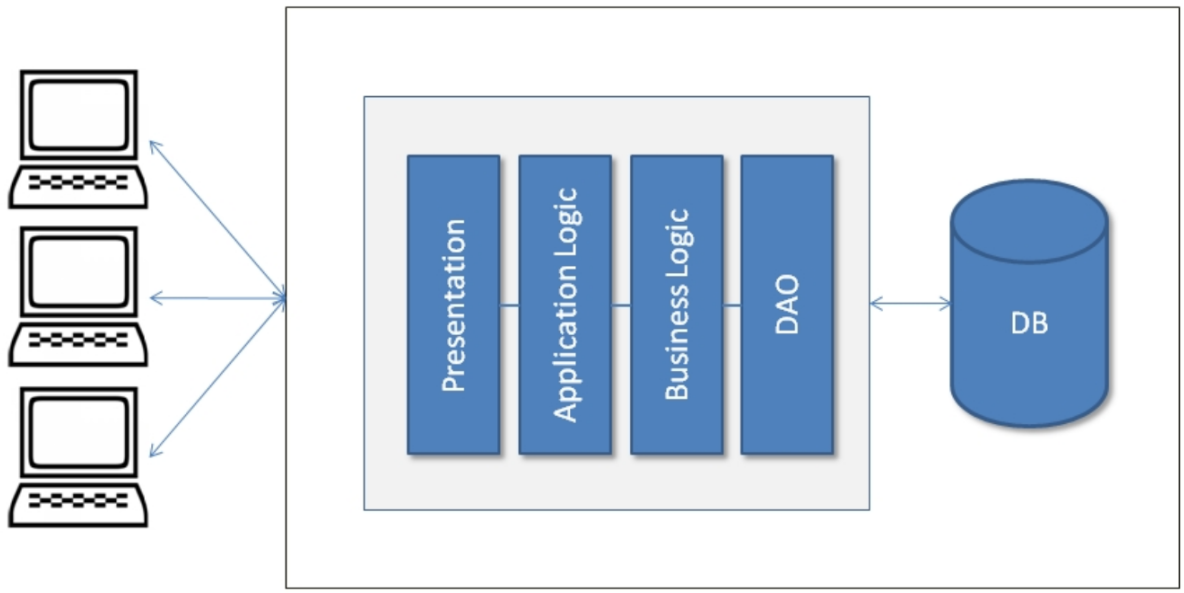


Figure (3) – example of monolithic architecture system

Monolithic architecture is based on a single executable artifact, called monoliths, which is crafted in languages like Java, C/C++, and python. Modules created with these languages are dependent on each other when it comes to execution, development, and deployment making resulting in a software application (Dragoni, et al., 2017). Figure (4) shows another type of a monolithic system. (Barashkov, 2018)

In their paper, Kratzke (2018) said that the application under the monolithic architecture has to be deployed all at once; this includes updating or modifying a service, nevertheless, adding a new one. Moreover, they described a case where one complete application could be packaged as one large machine image. In this case, difficulties are faced when the application's size is relatively

large, causing a downtime for users or customers using the application and constraints the application's scalability.

Applications based on monolithic architecture can grow over the years, making them complicated and hard to maintain. With the real effort put into the application's development, it does not become efficient anymore to have an application where a developer cannot keep insight into all components. Monolith applications are known to be scaled by duplicating the application instances rather than on the module level. (Fritzsch, Bogner, Zimmermann, & Wagner, 2019)

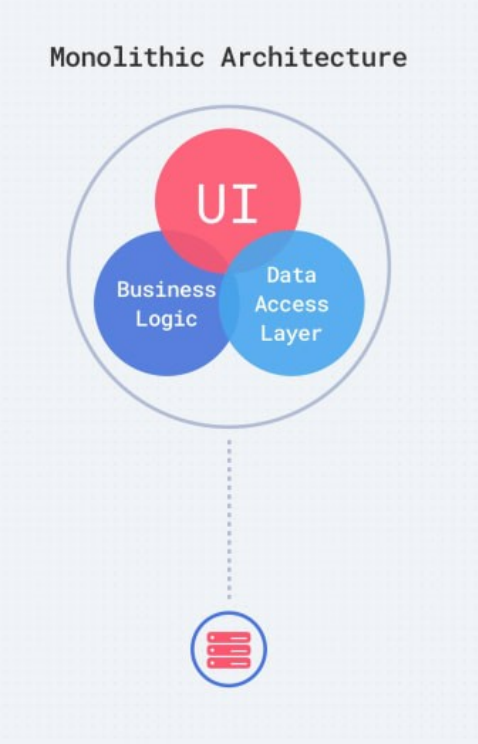


Figure (4) – Another example of monolithic architecture

The application was built using the monolithic architecture have strengths. Firstly, the functionality concerns one application resulting in fewer concerns in the logging, handling, and monitoring. Secondly, debugging and testing are end-to-end and fast since it is one single unit. And lastly, monolithic architecture is simple to develop and deploy; just one directory and a knowledgeable team are needed. (Gnatyk, 2018)

4.3. Comparing Microservices to Monolithic Architectures

4.3.1. Availability

With service availability, monolithic applications are one single code that functions together. If a part of the Monolith is updated, fixed, or modified, the system becomes unavailable for the period. This, in return, affects the scalability and maintainability of the system using a single technology development. (Sarsansig & Leon (2019), Tapia et Al. (2020))

However, microservices architecture, as a distributed system that has all services working independently, increases the availability of the system. Independent deployment of services allows the rest of the services to stay available for use in the meantime and making the services more fault-tolerant since the necessity to update the whole system is not present. (Tapia, et al., 2020) Microservices should also be characterized by differentiated availability depending on each service or part of the system (Balalaie, Heydarnoori, Jamshidi, Tamburri, & Lynn, 2018)

4.3.2. Security

Security is one of the main challenges that face microservices and Service Oriented Architecture. Dragoni et al. (2017) discussed in their paper the main security issues. Firstly, compared to monolithic architecture, distributed services that communicate through the network are more prone to being attacked than a single constrained OS. The second challenge is the network complexity; provided that there are many small independent components, it might result in complex network activity. The complexity can further make the security protocols harder to apply, resulting in more challenging monitoring and auditing. Attackers are then making use of the complexity of the network to perform attacks.

Moreover, trust is one of the main concerns where an attacked service can control and maliciously affect the rest of the distribution as they are communicating together. It is recommended for microservices to have a mechanism that confirms security within services. Lastly, heterogeneity is the last concern since the microservice system enforces heterogeneity between different systems, allowing different security domains without a common security infrastructure. This results in a system that does not have a globally enforced set of rules to control security.

Therefore, the monolithic system seems to be more secure and safe to use. On the other hand, learning and understanding the challenges that would face a distributed system could allow for a solution to make it more secure.

4.3.3. Testability

Different services communicating together are evolving at different paces. In other words, a system that is made of one big code is being tested in the same environment, being manual or automated. While for microservices, creating a consistent environment for different independent services, integration testing can be challenging. (Dragoni, et al. (2017), Nielsen (2015))

Each component in microservices is therefore tested independently or in isolation. That being said, it allows testers to isolate services and adjust the scope of the test. Moreover, it allows including

services based on the scope of change; either which has been modified or remained unchanged . (Dragoni, et al., 2017)

4.3.4. Database

The microservices architecture database is designed to be in a container alongside the rest of the services communicating. The service or the host accesses each database, there also exist another container that keeps the data generated synchronized in order to prevent loss after the execution of any container. In this architecture, multiple databases could exist which already exist in different containers. (Stubbs, Moreira, & Dooley (2015), Tapia, et al. (2020))

Both types of architecture can store data in similar databases, such as relational databases, as Villamizar et al. (2015) showed in their comparison. Meanwhile, a monolithic architecture shares the same database with all the services that exist in the same codebase. In this case, the database is not independently executed and part of its shared resource. (Dragoni, et al., 2017)

4.3.5. Reliability

System reliability is defined as the probability of the system performing its tasks properly in a determined period, without the need for repair or changing system predefined specifications. (Muhammad Raza, 2019) Decoupling services increases each service's reliability, individually, but at the cost of the application's reliability as a whole, as Omondi (2019) explained.

Omondi further explained reliability in monolithic systems, where one issue with the network, memory, or something else, can cause the whole system to go down. The downtime of one service as a whole, Monolith, is the provider's downtime, being 0.5% or 1% of the time. While for microservices, the downtime is the number of services multiplied by the downtime provided.

Therefore, microservices require more measurements to deal with potential application failure. When one service fails, the rest is still working. Moreover, Dragoni et al. (2017), in their paper, supported this by stating that every distributed system has inferior reliability compared to applications using in-memory calls.

4.3.6. Development

Developing a Monolithic system is one where developers have been developing for years. Fritzsch et al. (2019) mentioned in their paper that developing an entirely new microservices-based system is time-consuming and very costly. On the other hand, an attempt to transfer a monolithic architecture to microservices can also be time-consuming, depending on the system in hand.

The size of the system has a significant impact on the way the application should be developed. In other words, as mentioned earlier, microservices can benefit best a system that has an extended code with many services. Whereas monolithic architecture is best suitable for developing a system to be used by a small number of users and providing a relatively small number of services. N. Dragoni et al. (2017) discussed the size of a system. Provided that service is too large, splitting the service would be ideal in terms of maintainability and extendibility.

It is safe to conclude that adding more services and new features to a microservices architecture is less complicated than with a monolithic one. Developing a monolith means re-implementing a whole system for every change. Daya et al. (2015) supported this by mentioning that every service in a microservice has the availability of being developed in its language, independently, which facilitates adding new services and features to the existing system.

4.3.7. Performance

Flygare and Holmqvist (2017) presented a bachelor thesis about performance characteristics between monolithic and microservices systems. The study included an experiment on two systems from both types: a comparison of the latency, successful throughput, the RAM and CPU measurements.

The result of the experiment had a conclusion that was similar to that of Villamizar et al. (2015) that the number of users is a determinant of the microservices' speed. In other words, microservices can be slower in some contexts but somewhat depending on the companies using the system. Similar findings have also been concluded by Al Debagy and Martinek (2018), stating that monolithic provides better performance for a small load of users than microservices. They further concluded that a monolithic architecture's average throughput number is higher than the one concluded from the microservices system. However, they compared two different microservices based on their service discovery technology and found a difference in the performance.

Hence, this section concludes that both microservices and monolithic architecture's performance depends mainly on the number of users and the technology used for the implementation. Moreover, using dockers to containerize a system increases the performance on the aspects of RAM and CPU (Flygare & Holmqvist, 2017)

4.3.8. Scalability

Building an application that has an excellent usage potential means that later on, more users will be drawn to the system and more expectations will be set accordingly. If an application cannot be scaled for the usage, the performance is then negatively affected, and business users might complain.

Scalability is the process of growing the application to meet the users expectations without delay. The application is then capable of handling more requests per a set calculated time. Scalability depends on both software and hardware, and the change can be, for example, increasing the memory limits to increase the throughput. (Prasad, 2019)

When it comes to microservices, the architecture has the advantage of scaling each element separately, so it is considered to be more time-efficient than it is with the monolithic, needless to say, cost-efficient. On the other hand, with a monolithic architecture, companies might end up rebuilding their Monolith. With new users come new problems, and therefore scaling a monolith, which needs to scale the whole application and not just the needed part, resulting in limits within the scalability. (Gnatyk, 2018)

4.3.9. Maintainability

When a code is well written and understood, microservices can have increase maintainability. Being a distributed system with loose coupling between services, fixing, modifying, or adding a feature to service costs less than monolithic. (Dragoni, et al., 2017)

Tapia et al. (2020) saw that maintaining new issues using microservices can be time-consuming next to migration. When the system is smaller, monolithic systems are seen as a well-maintained solution, depending on usage. Staying with a monolith could be easier to manage, but if microservices are integrated correctly, the application can profit from innovation capacity.

4.3.10. Table of comparison

The information gathered above in this section is summarized in this section in a table of comparison between the two architectures highlighting the main points of differences.

Point of comparison	Microservices	Monolithic
Availability	Microservices are independently deployed increasing the availability of the service	Having one single code relatively decreasing the availability of the application
Security	There exist several security challenges compared to traditional systems that need to be taken care of to maintain the system	Relatively presenting fewer challenges but also require typical security concerns.
Testability	Testing each component can be faster independently, while integrated testing can be challenging	Testing one big system can be easy but time-consuming
Database	The database exists in containers that are communicating with the rest of the microservices for the prevention of data loss	A centralized database that is connected to the main codebase of the application
Reliability	Microservices increase the reliability of each service independently	The whole application shares the same reliability, and any change affects the reliability of the whole system
Development	Development and deployment of the services are available, easy and cost and time-efficient.	Development takes a significant amount of time and resources. The whole application needs to be deployed in order to change one setting.
Performance	Provides relatively better performance when the number of users is high	Provides better performance for a small number of users
Scalability	Each service can be independently scaled, which makes the scalability easier and cost-efficient	Scaling is very hard as the need to scale means scaling the whole application

Code Language	Each service can be developed and deployed in its coding language	The whole application is written in one code language.
Maintainability	Fixing, modifying, and adding new features cost less time and effort for a distributed system	When the system is small, maintaining a monolith is more manageable than when the system gets bigger, where it becomes more complicated and harder to understand

Table (1) – comparison summary microservices and monolithic

4.4. Banking Sector

Technological advancement are one of the main focuses of any enterprise that wants to keep up with world trends and stay on top of their sector. The banking sector is no different from any of the other sectors, and it might even be more crucial in that sense. In today's world, especially with the significant reliance on smartphones and online tools, the banking sector offers its services online and in between the user's hands by a few clicks. In America by itself, there has been an increase of 20% of users from 2014 to 2019, resulting in more than 160 million online banking users. (I. Mitic, 2020)

Earlier in 2020, Juniper Research published research results on the digital bank predicting the number of digital banking users to exceed 3.6 billion by 2024, which is a 54% increase from 2020 with 2.4 billion users. Moreover, the online banking sector has been taking over customer acquisition because of the advanced user experience offered compared to traditional banks (Juniper Reasearch, 2020). This is clear motivation for well-established banks to earn their place back in the market by working on their Unique Selling Point (USP).

Campanella, Della Peruta, and Del Giudice (2017) studied the effects of technological innovation on the banking sector. Seeing that the way banks used to be changed and that the product range, service types, and developments have changed, they concluded that with the increasing change in the world Information and Communication Technology (ICT), there had existed some innovations that have positively affected the earning margins of banks. Instead, there is a general influence on the banking system's competencies, abilities, and organization.

This all shows how the banking sector requires technological advancement, and banks need to be doing their utmost to stay on the safe side of technology. It seems that the number of users in online banking services is increasing notably. The increase can be, without a doubt, good for the image and the profits of the bank, but as explained in the previous sections, the traditional, monolithic architecture could be hard to maintain with a large number of users cause a problem for scaling the business applications.

Very few papers concentrate on the transformation to microservices architecture from monolithic in the banking sector. However, a considerable case study that has been published was based on the Foreign Exchange (FX) core system of the Danske Bank, the largest bank in Denmark and one of the leading financial institutions in Northern Europe. Mazzara et al. (2018) and Bucchiarone et al. (2018) wrote about the bank's experience and the transformation of their FX Core system. The project has improved the banks' scalability through microservices as stated by Bucchiarone et al. (2018)

The FX IT system at the Danske bank is part of its Corporates and Institutions (C&I) department, the gateway for communication between the bank and international markets. The FX Core system is the part of the FX IT that checks the customer's collateral to perform a trade and its effect on such collateral (Bucchiarone, et al. 2018). The system was already in the service-based direction where the services were deployed individually and across a cluster. The system further used APIs

for interfaces of the clients and messaging services for external providers. Figure (5) represents the FX core system's old monolithic architecture (Mazzara, et al., 2018). However, as the system shows a possibility for scalability, the bank encountered problems trying to rapidly and consistently develop the system and deploy changes (Mazzara, et al., 2018)

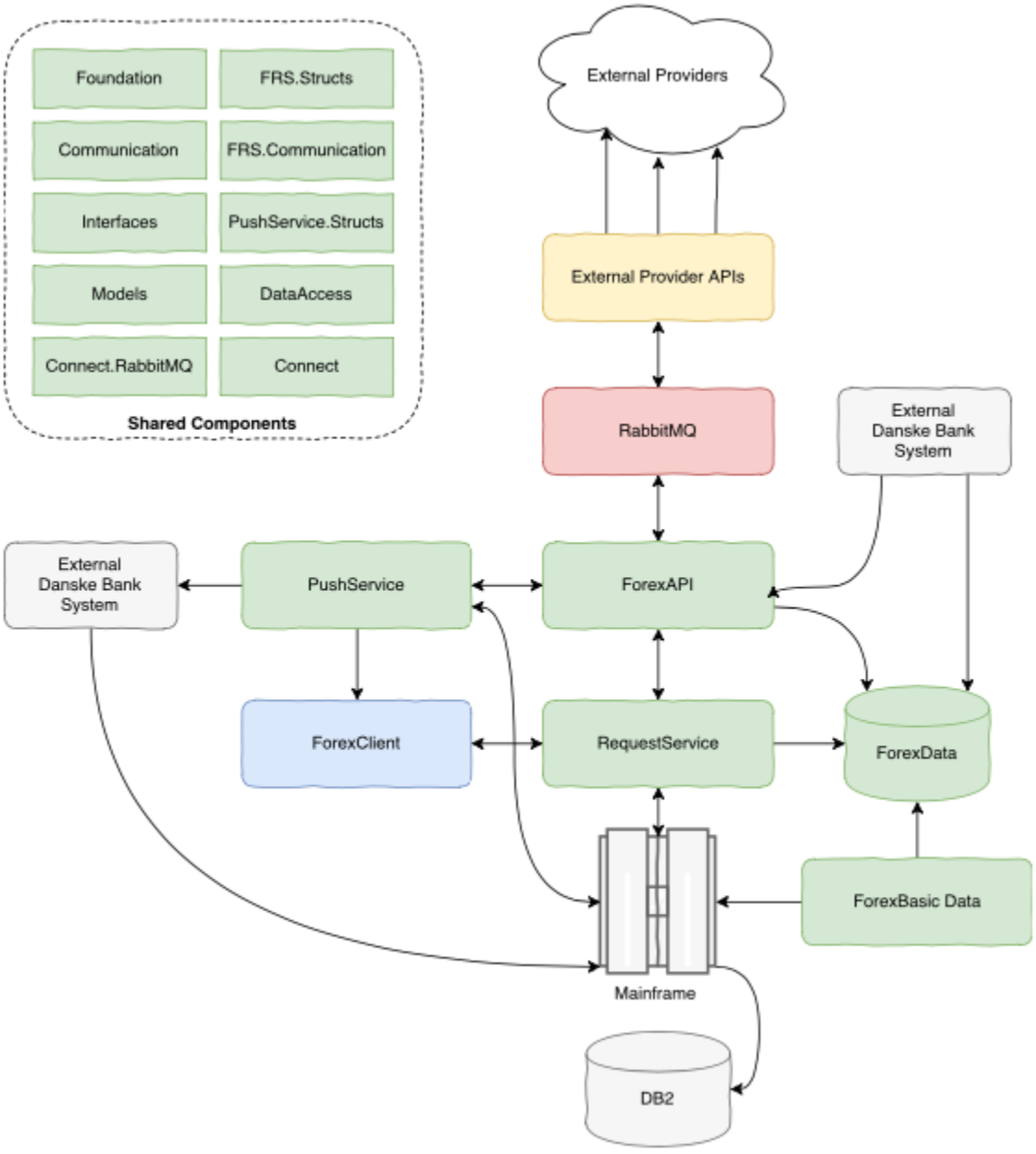


Figure (5) – FX core system's monolithic architecture

Furthermore, scalability was not the only problem that faced the FX core system and was taken into consideration. Mazzara, et al. (2018) explained the rest of their paper's problems, starting with too big components, causing complexity and confusion. Moreover, since many of the components were shared among the services, modifying one of them resulted in updating all services and testing

all dependents, leading to unnecessary coupling. The mainframe was seen as another problem for being developed in older legacy technologies, and a lousy overview of the system casing difficulties optimizing the system.

In continuation of the problems that were caused by the monolithic system, the most common, or mostly the reason why monoliths are most problematic, was the complex deployment caused by high coupling. Rebuilding and redeploying the system was mainly needed after a single change along with testing. Organizational culture was the next problem where people were not educated enough in software architecture, causing the team to be unable to modify the database structure and breaking essential business processes. The various integrations in the system, further needing different communications, can cause complexity and higher coupling, as explained in the paper.

The last two problems discussed in the FX core system's monolithic system were technology dependence and system status overview. The first corresponds to the limitations set by a monolith to develop in the technologies implemented, even if the choice of technology for the development is not the most efficient. The latter corresponds to the manual check for logs, messaging systems, and the lack of a preventatve system warning, this is caused by the inability to check an overview of the system because of the decentralized location of the system.

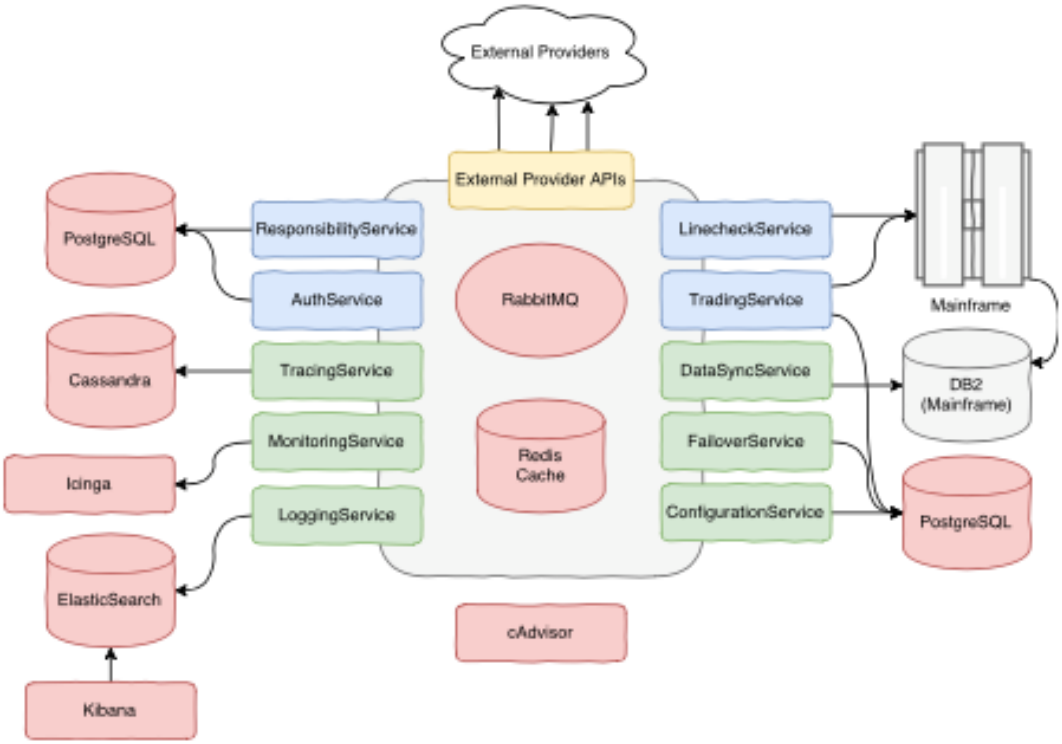


Figure (6) – FX core system’s new microservices architecture

The new FX core architecture is a microservices-based one that is intended to replace the monolithic. Figure (6) represents the new microservices-based FX core system (Mazzara, et al.,

2018). Bucchiarone et al. (2018) explained the architecture as an example of change implemented in an enterprise setting. The main change in the architecture goes as follows – briefly explained and furtherly discussed in the paper - Containerizations/Dockers enable Docker tools, such as automatic deployment. Automation, where all services have an automated Continous Integration and Continous Deployment CICD pipeline. Orchestration allowed by Docker tools helps failed services to restart automatically and allows service discovery and load balancing. Lastly, services integrate via message passing choreography.

The new architecture helped to solve problems by creating small independent services that are loosely coupled. Furthermore, only one shared component existed – lambda frame – a frame to connect infrastructure and provide standard formatting methods. Development now is becoming comfortable with less limitation due to the independence of the services. The team now has higher control over the whole infrastructure; they can develop open APIs for their clients and traders. They also have no technology dependency in architecture and can integrate new technologies and languages. (Bucchiarone, et al. 2018)

This story has shown potential in solving problems in the banking sector when the team is dedicated and willing to change the system to a more understandable and adaptable one. Microservices have benefited the Danske bank, but there will always remain the need to revisit the services, the functionalities, and the business policies. This case should be a good start to investigate further the possibility of decomposing another monolith into microservices in the banking sector.

4.5. Containers and Virtual Machines

This section is mainly focused on explaining the concept of containerizations and Docker. Throughout the research, these two terms have been used frequently, explaining how they come to use in creating a microservice architecture and how they can benefit the system and its developers.

Stubbs, Moreria, and Dooley (2015) introduced their paper by discussing how these two technologies significantly impact distributed systems and cloud computing. The trend for the usage of containers started growing away from the singly run software, monolithic. Since microservices are naturally a loosely coupled architecture, containers are considered an excellent match to achieve better modularity, code reuse, reproducibility, and better scalability.

First, containers have been widely present over the past years, which is mostly favorited because when introduced to a shared environment, it allows for complete isolation for software applications. Figure (7) shows a container environment (Bauer, 2018). By isolating the network between containers, file systems and resource consumption, container technology helps prevent traditional attacks (Stubbs, Moreira, & Dooley, 2015). Containers are considered a solution in migrating from one computing environment to a new one. Build into one package; containers contain resources and dependencies needed by an application neglecting the differences in operating system distributions. (Rubens, 2017)

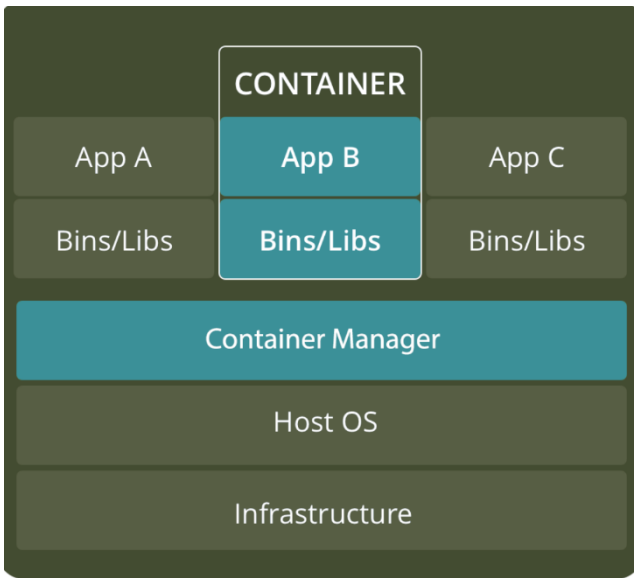


Figure (7) – container environment

Virtual machines are similar to computing environments; Figure (8) shows a virtual machine environment (Bauer, 2018). Virtual machines are explained as a matched reality of an actual computer functioning normally to execute programs. Virtual machines run on top of a physical machine using a hypervisor – a piece of software, firmware, or hardware – which in return runs on a host machine – providing the virtual machine with distributed resources. (Kasireddy, 2016)

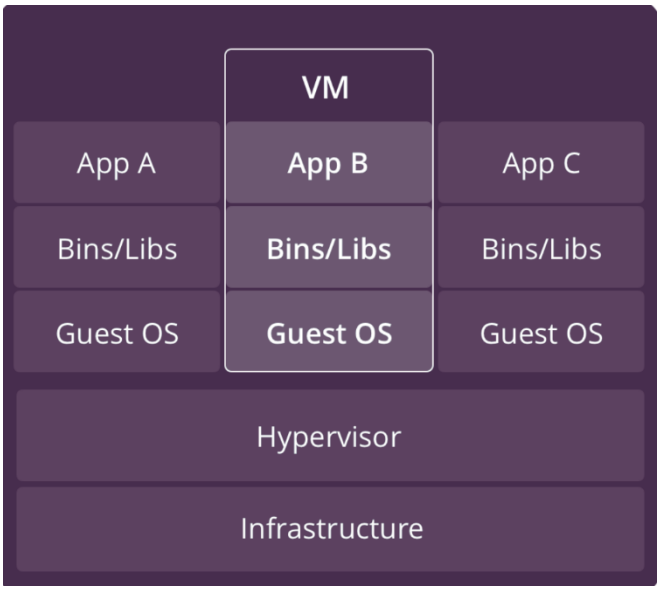


Figure (8) – virtual machine environment

One of the main reasons microservices are showing success as architecture is the ability to standardize services in containers. Making use of the operating system virtualization where containers share the host system's kernel with other containers (Kasireddy, 2016), not the same as a virtual machine, being more lightweight and using fewer resources (Kratzke, (2018), Stubbs, Moreira, & Dooley (2015). The containers are, therefore, useful for faster and more pragmatic scaling within the microservices architecture. (Kratzke, 2018)

Furthermore, Docker is described by Stubbs et al. (2015) as a natural evolution of the container technology integrating and extending container technology providing a single lightweight runtime and API to manage the execution and managing the containers and their images. Compared to a standard hypervisor, Docker allows the application to prepackage its dependencies resulting in less needed overhead to run. The main reason why microservices is an excellent pair to Docker is its ability to create complex systems across the different environments with the need for fewer resources and providing a faster turnaround than what is known.

There are four reasons stated by Kasireddy (2016) in their article why Docker is gaining momentum in the field. Firstly, Docker can be easily used by admins, developers, and others, to build an application and run it on any type of cloud, "build once, run anywhere." Speed is the second reason where Docker is very lightweight and takes fewer resources. It takes seconds to create and run a Docker container. Thirdly, with the least modifications needed, Docker offers a full hub for users to pick their images from, readily available to all the users as an ecosystem. Lastly, the modularity and scalability of Docker help breaking down the functionalities into separated containers.

Table 2 is therefore introduced with the key differences between virtual machines and containers to summarize the above mentioned. (Bauer, 2018)

Virtual Machines	**Containers**
Heavyweight	Lightweight
Limited Performance	Native Performance
Each VM runs in its own OS	All containers share the host OS
Hardware level virtualiztaion	OS Virtualization
Startup time is minutes	Startup time in milliseconds
Allocated required memory	Requires less memory space
Fully isolated and more secure	Process-level isolation, probably less secure

Table (2) – comparison summary virtual machines and containers

4.6. Choreography & Orchestration

As discussed earlier, microservice architecture is about having a combined system of small services to solve a real business need. The whole system communicates together and completes business functions. With distributed architecture, communication is a tricky and an essential part of understanding. Therefore, this section will discuss service compositions, classification approaches, and methods of service compositions.

There are three categories of service composition algorithms: Non-heuristic, accurate and exact, and optimal solutions. Heuristic algorithms, which are based on the methods of trial and error; does not provide optimal solutions but rather fast results. Lastly, the Meta-heuristic algorithm, used for generalized solutions and is broad ranged. (Singhal & Chelliah, 2019)

Furthermore, orchestration and choreography fall under composition methods for services. Compositions are used to combine microservice units to perform the whole goal of the complex system.

First of all, visualizing orchestration in real life makes it easier to understand the composition method. There is an "orchestrator" or a controller centered on the interaction of the services altogether. Since the orchestrator is responsible for communications, a message with a request or response is followed by this composition's interactions. Figure (9) is an example of a service orchestration (Mat'sela, 2019). In orchestration, a response from the called service is expected to be received before calling the other service. The orchestrator is then responsible for sending and receiving the requests or responses for the services. (Singhal & Chelliah (2019), Rudrabhatla (2018))

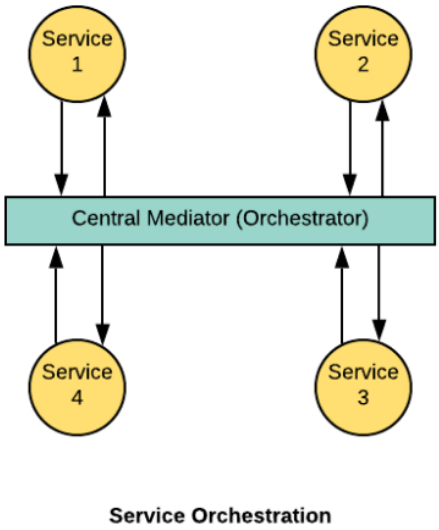


Figure (9) – service orchestration

The main benefit of having orchestrated services is making sure one service is done before the other needs to start its job. However, by doing so, the services are dependent on each other and coupled. If a service experiences a problem, the dependent services will do as well. Moreover, when the orchestrator is facing a problem himself, the orchestra cannot coordinate, in other words, single-point failure; if the coordinator fails, all processing and the application fail. The last trade-off, when employing orchestration, is the extra handling time it needs for execution. This means that if three services have to be executed sequentially, the time needed to happen time is S1 + S2 + S3 in order. (Singhal & Chelliah, 2019)

Instead of having an orchestrated system that coordinates the services, another possible technique is coordination or choreography. Singhal and Chelliah (2019) mentioned reactive architecture while introducing choreography, which brought up more curiosity about its relation. The Reactive manifesto, published online in 2014, explained the origin and was directed to by Jonas Boner in his book in 2016 explaining Reactive Microservices Architecture.

The Reactive manifesto (2014) explained that reactive systems are more flexible, loosely coupled, and scalable, making them easier to change and develop. With useful interactive feedback provided to the user, reactive systems are more prone to failures and highly responsive, as their name should suggest. This seems to be more of an explanation of how a well-built and maintained microservices architecture should be.

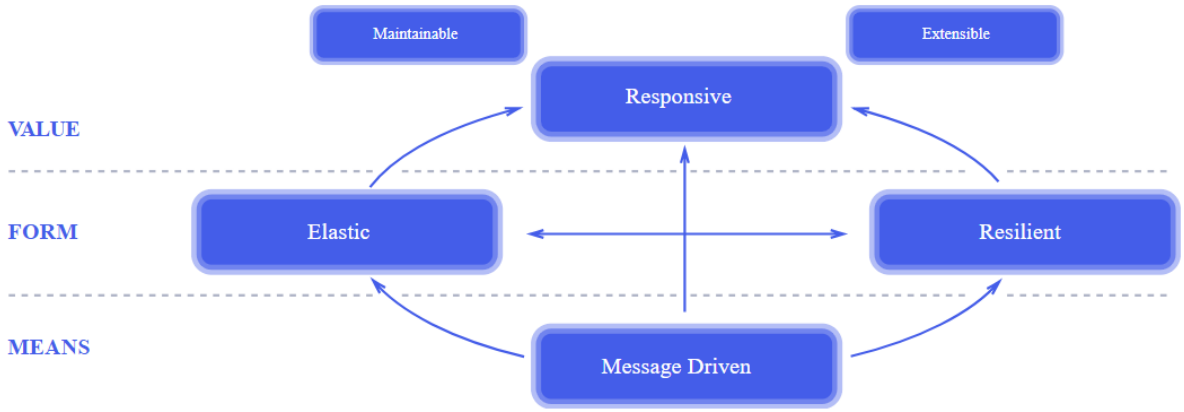


Figure (10) – the reactive manifesto

As mentioned earlier, Boner (2016) used the Reactive Manifesto as a foundation for his book. He explained that microservices architecture is an architecture that has learned from its previous boomer, Service Oriented Architecture. He described microservices as one of the most interesting reactive architectures in recent years of his book. As the manifesto explained, reactive systems are responsive, resilient, elastic, and message-driven. Figure (10) The Reactive Manifesto (2014)

Getting back to choreography, the reason the reactive manifesto was mentioned. In choreography, there exists no central 'orchestrator' controlling the logic in the application; instead, the logic is built into the services that coordinate together. Figure (11) (Mat'sela, 2019) A service publishes an event; the needed reaction is known ahead of time by the other services to trigger them. All services work autonomously; multiple services can respond to one event and work on their event back. In this case, the events keep happening until the last service does not publish any more events, and hence comes the end of the task or the transaction. (Singhal & Chelliah (2019), Rudrabhatla (2018))

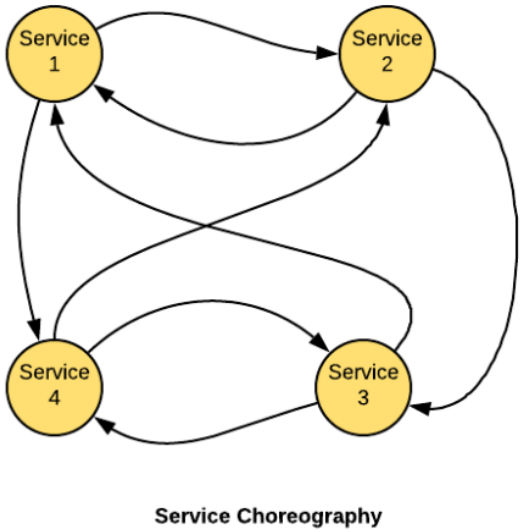


Figure (11) – service choreography

Service choreography benefits the system with few advantages that Singhal and Chelliah (2019) explained further. They started because without a central controller, services can be executed parallelly, which results in faster processing. Moreover, adding and removing services or updating them becomes more straightforward, focusing on each service on its own instead of the whole system, which aligns well with the agile delivery model. The lack of a central controller also provokes a single point of failure. On the other hand, as discussed earlier, this type of programming can be challenging for developers and their working mindset. Lastly, complexity based on each service having its logic and flow can be another challenge in this composition.

Orchestration	**Choreography**
Centrally controlled	No central control
A single point of failure exists	Single point of success
Coupled and dependent	Less coupled and dependent
Dependencies make it less easy to modify services.	Can easily modify the services
Services work based on a response from the orchestrator	Services work parallelly
It takes a long time to execute fully	Faster execution due to parallel execution
Less complicated and more common	More complex and challenging for some developers

Table (3) – comparison summary orchestration and choreography

5. Literature Review

This section of the study focused on the state of the art and existing cases that studied microservices in the business world. The literature is divided into three sections; the first section tackles the companies that implemented the microservices architecture and managed to make use of it to fix their architectural problems. The second is focused on the implementation of the technology and the design of the architecture. Lastly, the third section is about the companies that have failed to implement the technology.

5.1. Successful cases

One of the essential related studies to this thesis is the re-implementation of a monolithic architecture system to microservices in the FX Core system of the Dankse bank. This case represents a successful transformation in the financial sector and one of the sector's leading institutions. The case study aimed to focus on a repeatable migration process characterized by legacy systems and batch-based processing on heterogeneous data sources. (Mazzara, et al., 2018)

Uber is one of the companies that moved from monolithic to microservices to solve problems that arise from expanding the application. Worldwide. Problems included the need to rebuild and deploy the features continuously, the necessary modification of the code when fixing bugs, and the difficulty of introducing new features or scaling the application. By introducing an API gateway connecting all drivers and passengers and transferring the services into individually deployable ones, Uber successfully moved to microservices. (Kappagantula, 2018)

Daya et al. (2015) wrote a book where one chapter focused on using IBM Bluemix to transform CloudTrader; A java platform that stimulates an online trading system, as a case study. Using auto-scaling, Session Cache enables persistence for all sessions in case of failure, and SQLDB as their primary database, manages to transform CloudTrader to microservices. The transformation results provided a new component for each service written in its language and more conducive to constant change and increasing the agility of delivering new features.

5.2. Design and implementation

Wix.com had a journey moving from a monolithic architecture to microservices that began in 2008 by breaking down the Monolith in 4.5 years. Moreover, wix.com needed two years to complete the transition to microservices and built its microservices framework. They had to invent new testing and integration patterns and realized four main groups of services. One group responsible for hosting and serving published sites supports creating a website, an internet media filesystem, and a set of applications that add value to Wix sites. (Yoav Abrahami (n.d), Wix Engineering (2015), Bill Doerrfeld (n.d))

Companies can need to change the tools used; CloudElemets described their journey to discover that required design and debate. (Bill Doerrfeld, n.d.) CloudElemets are using Docker, like different companies who switched to microservices and running them remotely. Docker is

container-based virtualization software that has minimal impact on processing, memory, and network. (Al Debagy & Martinek, 2018)

The design of a microservices architecture requires an enabler for small or medium-sized services, and this is why Stubbs, Moreira, & Dooley (2015) discussed the usage of a container. They also mentioned the need to avoid a complicated network topology. Orchestration is also an essential aspect of designing a distributed system; Nielsen (2015) mentioned that it is introduced to the system as an independent microservice. They also mentioned, along with Villamizar, Castro, Merino, & Casallas (2015), the need for lightweight architecture and consumed fewer computing resources.

5.3. Failure Cases

As mentioned earlier, microservices can be a solution for many companies, but it is not a solution for everything. Stenberg (2014) wrote about a failure experience told by the Chief Software Engineering of Berico Technology. The reasons behind the failure were not technological, but they were stated as follows; Developers disagreement, service boundaries causing barriers like lack of focus on the main goal, independent services, choosing eight services instead of a couple to start with, and Implementing DevOps which took much time to learn.

Moreover, Steven Lemon, a Senior Software Developer, wrote about how his company canceled the move to microservices after starting a big bang project. Spending two months focusing on only breaking down the Monolith, the implementation of microservices did not only fail to prove its benefits from the implementation but caused more problems. The microservices' size was one of the main problems the team faced and caused many development issues. The team ended up splitting their solution into smaller projects rather than services and abandoning the idea of microservices. (Steven Lemon, 2019)

An article written by Mark Boyd talks about the Haufe-Lexware Group's CTO experience for nine months with microservices. Reinhardt, the CTO, declared the project's failure after noticing a long time needed to deploy a codebase change. Reinhardt further explained how separating user management had decreased the test coverage to zero; agile has become fragile and decreased the team's morale. The project's conclusion did not emphasize that the problem was with technology, rather communication, culture, and stating that microservices should be treated as a philosophy rather than architecture. (Mark Boyd, 2016)

6. Methodology

For this research, the focus is to understand microservices and recognize their aspects by gaining solid theoretical knowledge about the subject. This is followed by defining the research question, which will help develop a model to answer the question. The model is designed through a case study that is applied in a financial sector company.

Based on the understanding of the subject at hand and applying the knowledge gained through research and literature to the internship period, the conclusion is then used to answer the central questions of the research. The outcome of the case study will be used in conjunction with the research conducted to observe and discuss the findings for the thesis's use.

6.1. Research method

A literature review has been conducted systematically in the form of secondary research to be used for this master graduation project thesis. All the theoretical assumptions drawn in this research have been supported by existing theories derived from peer-reviewed books and papers and supported by real-life examples found online. Peer-reviewed documents and booked were accessed through different databases and were finely selected to match the study's analytical needs. The knowledge gained from different real-life companies is gained from company websites or blogs created by scientific researchers and workers in the field.

Qualitative research methods used in this thesis are observations and semi structured interviews. By taking notes during the internship period at ABN AMRO from the processes, about the data and the flow of work, an appendix should be presented by the end of the project to conclude the observations. Moreover, semi-structured interviews will be the lead of the observations and both shall be compensatory for the conclusion of the thesis.

Qualitative research has the advantage of being flexible and can be done in the natural settings. Meaning, data collection come from real work occurrences and new ideas and patterns are discovered.

Furthermore, the thesis will be developed based on a case study designed for a company in the financial sector. The aim of the study is to create a method to be used by companies in order to help them in the decision making towards microservices. Along the study, the method will be validated by presenting its different states to stakeholders and gaining insights from experts in the field. Choosing a case study as the preferred method for this research is based on multiple factors. Since the application of the technology is considered somewhat recent, the process of the transition is still in the exploration phase.

Moreover, based on the literature provided, some of the companies found problems in the transition specifically in their learning path. By introducing a case study in the field and taking some of the concerns into considerations, the study aims at providing a better understanding and a guide through some of the problems found.

6.2. Research design

The research design is often not clear in the beginning of the study. Through gaining direct and indirect knowledge of literature observation, the research question is then starting to get its final shape. Moreover, as current situation of the financial company develops over time with regards to external factors, the research question is defined. In other words, the collection of theory, research question and methodology, will result in the research design.

From the previous sections of the study, the literature review presents different cases of applying the microservices architecture in different business fields. The transformation is always accompanied by a lot of obstacles that are not only technical but cultural within the teams. In order to make the most out of the knowledge gained from the literature, it must be aligned with the finding to be discovered from the case study and the experts. The knowledge gained in the duration of the case study is then noted and organized in an academic approach to be further use it for the analysis of the findings.

Data collection for this case study is mainly based on meetings and following the flow of the team during the internship period. Some of the data collected are therefore through semi-structured and unstructured interviews with stakeholders of the project deducted in the financial sector. Accordingly, a participant observational case study design is chosen as a way to collect data and assure reliable insights by inserting the researcher in the middle of the environment. (Jhangiani, et al. (2017), Observational Research (2019))

6.3. Research approach

In order to support the research, an internship of nine months has been spent at a company in the Netherlands' financial sector; ABN AMRO a bank operating in Europe. The internship is taking place in Amsterdam in the IT PB&CIB department working in an agile environment.

This section describes the action approach to be used in the thesis where the researcher and the participants collaborate to link theory to practice. this shall help the development of the answer to the research question.

One way into making steps towards the research discussion, is by starting with the literature. By studying different cases from the business world, a definition of how to interview stakeholders is developed. From these semi-structured interviews, a list of requirements in terms of questions to be answered by the end of the project.

The interviews will be conducted in the form of meetings with people of expertise and stakeholders at ABN AMRO a bank operating in Europe. The internship takes place in Amsterdam in the IT PB&CIB department in an agile environment that helps with weekly meetings to keep the flow of the project going. The project is running under the name of architecture repository aiming to create transparency in the bank.

A list of questions has been presented after discussions to understand what the project wants to achieve.

- Where does the bank spend its money on change?
- What are the strategic key capabilities for the bank?
- Which grid is busy with what capability?
- What is the impact of legislation and regulations?
- Where are we using 'Software as a Service'? Etc.

The questions listed above were derived in the first steps of the project, these questions are then taken and discussed further to come up with more clear questions that direct the project into more definition. However, the team had decided on a specific approach to design the model that consists of five main steps. Figure (12) shows the approach that will be taken to build the graph meant to answer the questions of the project. Moreover, it illustrates the process with the goal of achieving an understanding of the IT landscape that could be later used for further analysis.

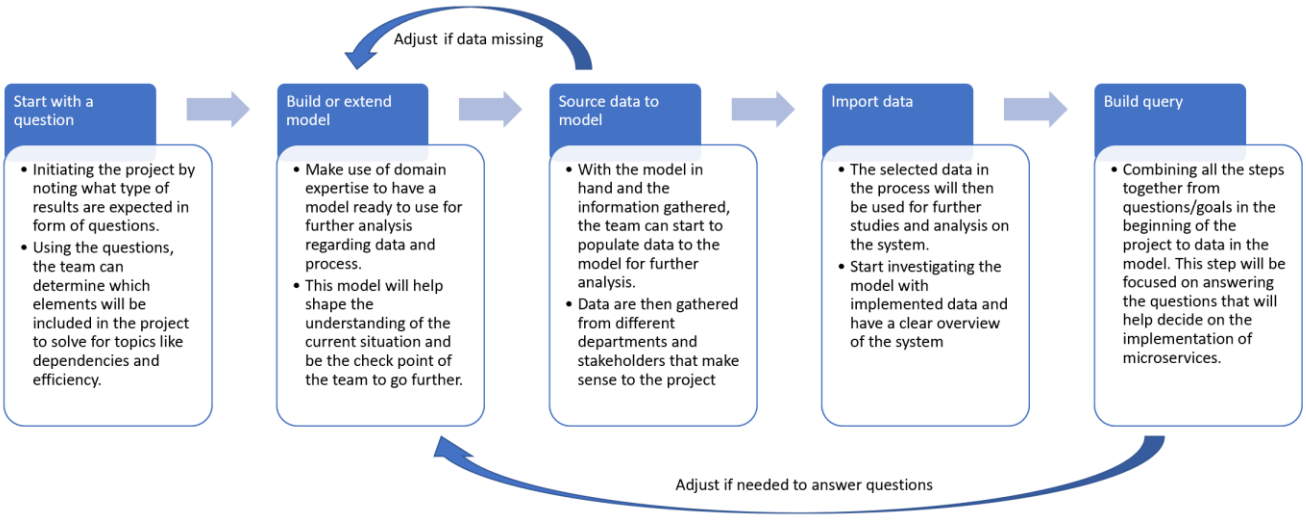


Figure (12) – steps to building the graph

As illustrated in the figure above, the team wanted to have an iterative approach towards few steps to make sure it is scaled in the right matter. The team is expected to achieve a milestone by each step that would help reach the goal of understanding the system and be able to do studies. These steps would further help the team achieve a level of clarity, especially when it comes to microservices architecture towards the end. Table (4) shows the expected results of each step which takes a part in the final goal of the project.

Start with a question	A list of questions that represents the goals and desired achievements from the project.
Build or extend model	A model that illustrates the IT landscape of the bank.
Source data to model	Gathered data from stakeholders and relevant departments to the first steps of the project and make sure they are ready.
Import data	The data gathered would be imported in the model and used for further evaluations.
Build query	Combining all the previous steps to be able to answer the questions from the beginning of the project.

Table (4) – list of steps and their expected results

Moreover, after settling on an approach to be followed, another discussion took place with the aim of achieving a better definition of the questions proposed. another list of questions is picked and written down as a guide to what is expected to be achieved by the resulting design.

Which questions need to be answered by querying the database?

- Make clear about the responsibilities.
- Predictability, which parts of a system are affected by change?
- Impact - is the technology having a malfunction?
    - which processes are affected and how are they otherwise?
- Multiple servers for data attributes? in how many places does a piece of data exits?
    - how many versions of the data do we have?
- How are we calling elements?
    - different data to different database? mapping into the elements?

These questions are now taken as a guide for the team to know what they should focus on while working on the project. The questions were further represented in the following figure to make it clearer and easier to present (figure 13). The team considered the output as added value in terms of efficiency and effectiveness.

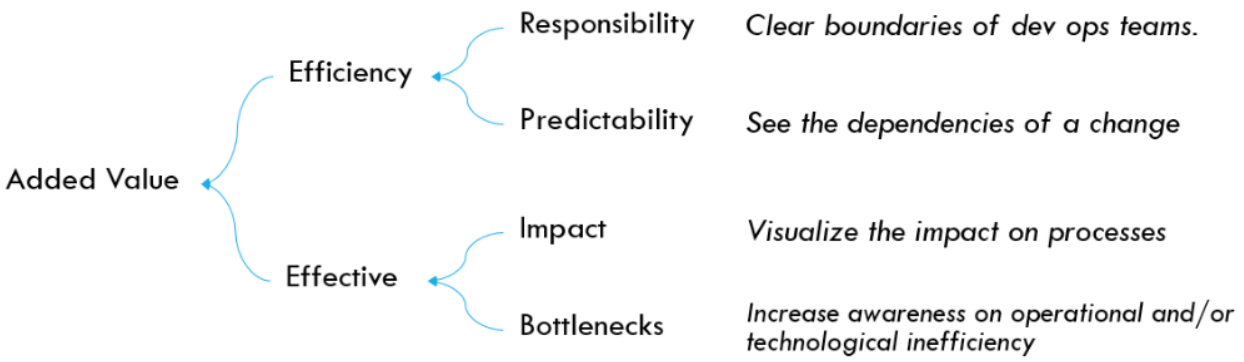


Figure (13) – breakdown of added value

Furthermore, answering the questions and achieving these goals have to be based on the visualization of the current state architecture. By considering the elements that exist, the team decided on the elements to be included in the design, this step will help the team in narrowing their focus down to have a clear idea on the parts of the system included in the study. The list goes as follows:

Application Component	Application Service	Application Interface
ARM function	BusinessProcess	BusinessArea
BusinessLine	EnterpriseParty	Contractual Relation
DataSource		

Table (5) – list of elements included in the design

6.4. Case study overview

ABN AMRO bank's case study is conducted in 2020 throughout the period of nine months, starting March 2020 till the end of the year. The team is mainly operating in Amsterdam, but due to the Covid-19 circumstances , the company is operating online, and employees are working from home. As one of the essential leading banks in the Netherlands with a department focusing on serving the whole bank's IT infrastructure and software services, a team has been gathered to work on a project that will positively serve multiple departments all around the bank.

6.5. Data collection

Throughout the internship, the team gathered every week to give an update on their progress and what their next step is. Documentation of the process has been a continuous goal of the internship and keeping track of goals and achievements. Moreover, further discussions with different team members gathered from different departments with different exposures have helped shape the case study included in this research.

The meetings conducted also included a few semi-structured interviews to gather more information with an end-goal of serving the thesis project. Due to time limitations and interactions that were

not doable unless it was planned, not all experts in the fields were interviewed; instead, the people involved directly in the project.

Moreover, as the team gathered, the discussions were always focused on the project and how to move forward. The team consistent of people of different roles and positions in the bank, however, not all team members came from the same department. The main department in charge of the project was as mentioned earlier, the IT PB&CIB department.

Employees involved in the process varied in their positions. To begin with, head of solutions and engineering was the manager of the project and the go to person. In the meetings, head of solutions would assign roles to the team and make sure that the main goal is being achieved through weekly (online) standups and check points.

Furthermore, the team consisting of an IT specialist an enterprise architect and a solution designer who had a huge impact on the development of the project. The IT specialists along with the enterprise architect had the role of investigating the current system and studying the possible softwares to be used in the process of modeling and presenting the data and the processes. The solution designer on the other hand, was responsible for the integration of the data in the model. Lastly, together the three were responsible for the testing of the implementation and detailed study of the system at hand.

Lastly, throughout the project, a few stakeholders from different teams would join the meetings to give more perspective about the current system. In these meetings, the team gathers more information about different processes in different department, gets more information about the data that needs to be implemented and receives feedback regarding the current progress.

7. Results

This section will focus on the research results and the period spent at the bank as an intern. Combining both the exposure gained from ABN AMRO and the knowledge gained from the research to derive a constructed results section will aid this research's conclusion.

The research conducted in this study was mainly to derive an understanding of the matter and find a way to answer the research question, finding a way to decide on the implementation of microservices to reach a beneficial result. The results can be broken down into main milestones that helped the team reach a transparent view of the data model and point out possible changes.

7.1. Case Study

By following the Eisenhardt framework (Maimbo & Prevan) a case study is developed and introduced for this thesis study. The case study is therefore done in three main phases: model development, model testing, and lastly model refinement.

7.1.1. Introduction

Established in 1991, ABN AMRO one of the most remarkable banks in the Netherlands and Europe. ABN AMRO is a merger between Algemene Bank Nederland (ABN) and Amsterdam-Rotterdam Bank (AMRO). The merge's primary goal is to create an organization that concentrates on strengths and able to scale the business up internationally.

ABN AMRO is organized in a manner that ensures high-class management and efficiency by the executives and supervisors. The critical elements of corporate governance and business operations are integrity, transparency, and accountability. The bank offers many channels for their clients to perform their products and services, such as a mobile banking app and internet banking.

With many channels and services in the bank, many datasets and tools need to be maintained and kept in good shape. Security, redundancy, and several other problems come into the discussion and become a concern for the bank's IT side. This case study is written to explain a project done in the IT PB&CIB department of ABN AMRO by a team dedicated to making the processes visible and transparent to the business side, project owners, and development teams.

7.1.2. IT PBCIB Department

Located in Amsterdam, the IT PB&CIB department is run and organized to deliver projects that serve the bank of ABN AMRO. This department delivers facilitated bank strategies through IT solutions for both Private Bank (PB) and Corporate and institutional bank (C&IB). The department is working in a community with different business lines, the Information Technology team, and other local teams throughout several countries. Cooperation with these teams results in delivering a portfolio of central and local solutions.

7.1.3. Transparency

The project in hand is handled by the Architecture repository team, defining transparency as a characteristic of seeing through; that is why the project is aiming to achieve. Transparency in IT PB&CIB is seeking to create a foundation for tactical and operational decision-making. The team

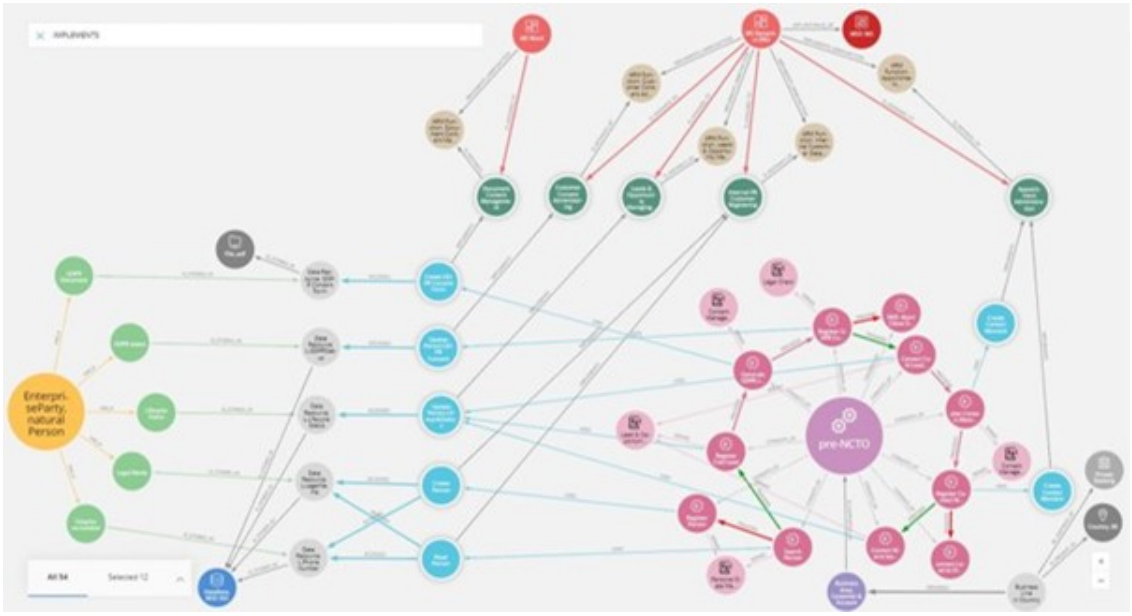
aims to add value in terms of efficiency by delivering clear boundaries for dev-ops teams and each change's dependencies. Moreover, project transparency wants to increase effectiveness by visualizing the impact on a process and increasing operational and/or technical inefficiency awareness.

The team gathered for the project starting by listing questions (Appendix) that would be asked to the database in the end. This database is created by modeling the process and linking the applications together. Later on, the model is used to implement different datasets that will be implemented in the tools existing to answer the questions.

The questions that have been proposed in the project, the team will be able to transform them into queries. These queries are expected to answer questions like the number of times a piece of data is stored and where, the impact of one technology on another, and the effect of one deficiency on the rest of the system.

The team managed to create the model (figure 14 & 15) and implement one of the datasets as a test to showcase the idea. By mapping the process into the dataset, a presentation to the managers will take place later. The project gained perspective and was approved to go further.

By answering the questions mentioned through queries, the visualization of the processes and the dataset can help the team identify dependencies and boundaries. The impact of such discovery can help the team to better maintaining and varying their infrastructure. One of the team's interests was to find a way to start implementing microservices into their systems. By identifying potentially replaced processes to be broken down, the team can decide whether they are ready to implement microservices.



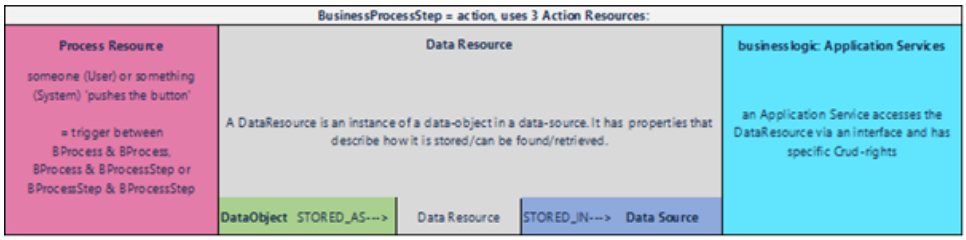


Figure (14 & 15) – model visualized and color breakdown

7.1.4. Visualizations

The internship helped shed light on the process of visualizing the bank processes and the data model that are in the study. One of the aims of the project is to be able to query the system to answer the questions mentioned above.

The transparency will show the way the data is flowing and will help identify potential API's replacements. This way, identifying microservices potential is easier to be open to discussion with stakeholders. The visualization process did not have the main aim of microservices but to have a clear view of the data, to begin with. This is a significant milestone indeed to reach a better understanding of the system before changing anything.

The visualization also helped identify alternative routes to different processes and enhancement suggestions that will help change and migrations.

Different modeling tools were used, such as ArchiMate and arrow tools. Using ArchiMate was beneficial in displaying different case scenarios and portraying the processes. Later on, as the data sources were identified and deployed into the arrow tool model, attributes were identified and used to aid in the querying process.

7.1.5. Development

An essential finding of the research and the internship is the team's dedication, and the culture within is a significant factor in this topic. As mentioned earlier in the literature review, one reason why a company failed to implement microservices was not the technology, but the team's culture and efforts put into the project to transformation.

This finding should be emphasized based on the case study that the team that was working on the project was a team gathered from different departments to achieve a goal, instead not dedicated to the implementation of microservices, which means that dedicating a team for this is necessary for the bank or any organization to have a success story.

7.1.6. Limitations

The IT PB&CIB project Transparency had a great initiative; the team started working from home as COVID-19 has hit the country. Work was slow, and progress was not as expected. However, after six months, the team managed to present the managers' findings and received feedback. The results were not exactly what they were expecting.

With the focus being distributed on different parts of the bank, the management team decided to merge the architecture repository team's effort to another project called clarity, done by the bank's business side and have a significant overlap with the aim of project transparency. Project clarity managed to identify three levels of data modeling: business, application, and infrastructure. Each level has its audience, purpose, and characteristics. Clarity has more focus on the bank's business side than the developers' side, as mentioned earlier for the IT PBCIB team.

7.2. Validation

This section is mainly about how the data collected for this thesis study is validated to ensure the credibility of the work done. Furthermore, the analysis done on the data is also to be validated by the method called triangulation analysis, which is based on the main results of the study (The case study and the research written.)

To begin with, since this study is prepared as a master thesis, all the sources used in the preparation of the research and the write-up has been documented and cited. Furthermore, the case study took place in the duration of the study period, during which the write-up is taking place.

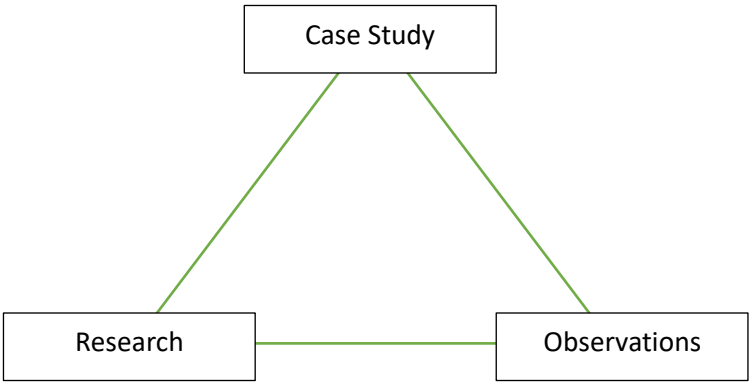


Figure (16) – data triangulation of the research

In order to make sure that the data analysis is done correctly, data triangulation is considered to be the most favorable method for this study. Data triangulation is the process of using more than one data source or approach to analyze the data at hand. This method is therefore used to ameliorate the credibility of a research study, hence provide a well-constructed analysis and results. (Salkind, 2010) Figure (16) shows the data triangulation method for this research.

Lastly the validation of this project will be confirmed after presenting the results to the stakeholders in the bank and the manager in charge. This is also to be confirmed by the appendix presented at the end of the thesis as a final report for the end of the internship period.

7.3. Data analysis

Is implementing an architecture like microservices that easy? And is it necessary to study the current architecture before doing so? How? The aim of this section is to answer these questions by analyzing the research done and the case study to provide useful observations for the findings of this research study.

Firstly, the literature review provided few cases where microservices were used in different companies such as Uber and the FX Core system of Dankse Bank. In these cases, the companies managed to operate the microservices architecture and succeed in their operational goals.

With a focus on the banking sector, The FX Core System is an excellent example of how an organization is focusing on one department or system to start the implementation of new technology. It is clear that a big bang is not the ideal way to implement a technology like microservices as it is not the easiest to deal with. This was shown in the literature where a senior software developer failed to complete the implementation as it caused more problems highlighting that the size of a microservice is significant to be determined.

Wix.com is another example that shows how long applying microservices can take long. It took the company over four years to make a complete transformation that included a whole two years of the actual necessary work. In the period of nine months spent on the case study present at ABN AMRO, the project was the first step into the study of the possibility of using a microservices architecture. This proves that this period taken to study the system is not long compared to other companies.

Moreover, a change in design might be needed just like CloudElements did in their transformation. In the period of transformation, problems will arise, and the teams will suggest alternatives. This was visible during the case study as well, where the team had suggestions for alternatives in the elements and relationships after drawing the processes.

Furthermore, many cases have not been successful in terms of transformation, and the main reason was not technical inability. For example, as mentioned earlier, Berico technology failed to implement microservices for multiple reasons, including starting with multiple different services instead of a small number and lack of focus from the team. This will be reflected later in the study since the lack of DevOps teams affected the current research and general progress.

Many changes will be expected in the testing environment as well. Since Haufe-Lexware Group also had to change their transformation after discovering that their testing environment reached zero coverage capacity. The team's morale decreased and the project was called off. This shows that the team's management of such a project plays a significant role in impacting the success rate and the culture projected.

Furthermore, the case study showcases a team in the banking sector focusing on reaching a transparent view of their infrastructure and their systems. The fact that the project focuses on seeing how the data is flowing and how the processes are working is a great base to begin with to

consider the microservices. The team worked in an agile environment, which made communication constant and documentation was a helping factor for sure.

With the processes and goals being documented, an ongoing process of mapping the processes happens, as shown in the appendix. However, the team organized was not a team dedicated to this project, so the goal was not committed. Meaning, as mentioned earlier, how many cases failed because of the team and culture, the case study does not showcase compliance in that sense.

On the other hand, since this project's path was mainly the start, it might be fine to consider the research or the study path with less dedicated personnel to clear the air for the decision making. This step is going to be very helpful in future practices of system decompositions or, as a smaller step, exploring the microservices potential.

As the appendix showcases the team's multiple steps to discuss, it is essential to take these steps and explain them with reflection to how these steps can help understand the system at hand. The next part will focus on these steps, explain how they were performed, and compare them to the research.

The team worked on providing a clear mission statement about how this project is providing help. One of the main points in the mission statement is to create a source for tactical and operational decision-making. This will happen through the modeling and showcasing of the relationship between the business processes and the information technology implemented.

This project should show clearly the boundaries of DevOps teams and the dependencies of the changes they can make. This is a critical point for the development of different technological implementations in general. And since microservices are based on small services that have to be independent, this will be an added value to the developers later on, in case the project continued to develop.

Furthermore, the department concerned will be able to develop more awareness towards inefficiencies, which can be seen already in the alternative routes suggested in the appendix. And by that, the last added value point will be achieved; visualize the impact on the processes. If the project succeeds in achieving these goals, decision-making in terms of microservices will be a doable mission.

However, before jumping to any conclusion, the project's main steps were split into five steps; stating the questions needed to be answered, building the model, sourcing the data to the model, importing the data, and building the query. These main steps were expanded and mainly based on research in order to be able to follow best practices.

The first step, starting with a question, was a step that was always updated as the team meet and think of more dependencies or more information that need to be understood. These steps were later followed by identifying key elements to be represented or stated in the model in order to correspond to the questions.

The questions were focused on the dependencies of the systems and what technology can affect another from working correctly. Further, it is essential to mention that also a focus was put on the database and how many times data is stored to make sure data duplication and dependencies are understood and tackled.

The second step is building the model; however, the team has to decide on their preferred modeling approach. Therefore, the model was drawn simple, and on a small scale (figure 17) and was discussed multiple times. The idea was to create a model that will enable the team to link applications to processes and technology and answer the questions stated in the project's first step.

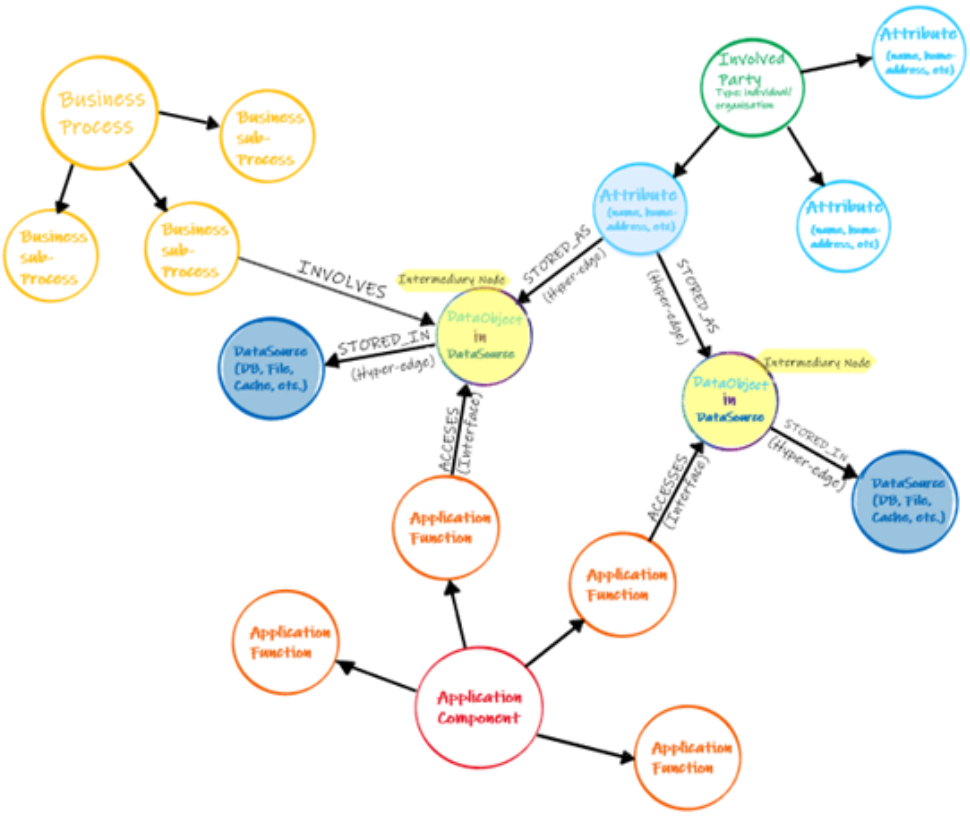


Figure (17) – small scale of designed model

The bank has its own technology and applications in use, which means compatibility with reading the data and importing it was a big concern in this step. In order to import different databases and test them onto the model, research has been done with different options that have resulted in the most suitable modeling choice.

The third step was to map a process into the model, which happens by deciding on a process to test and present to the management team as a reference to continue further with the process. This scenario is then taken into the fourth step, creating a database with the available technology that contains the relevant components to this scenario.

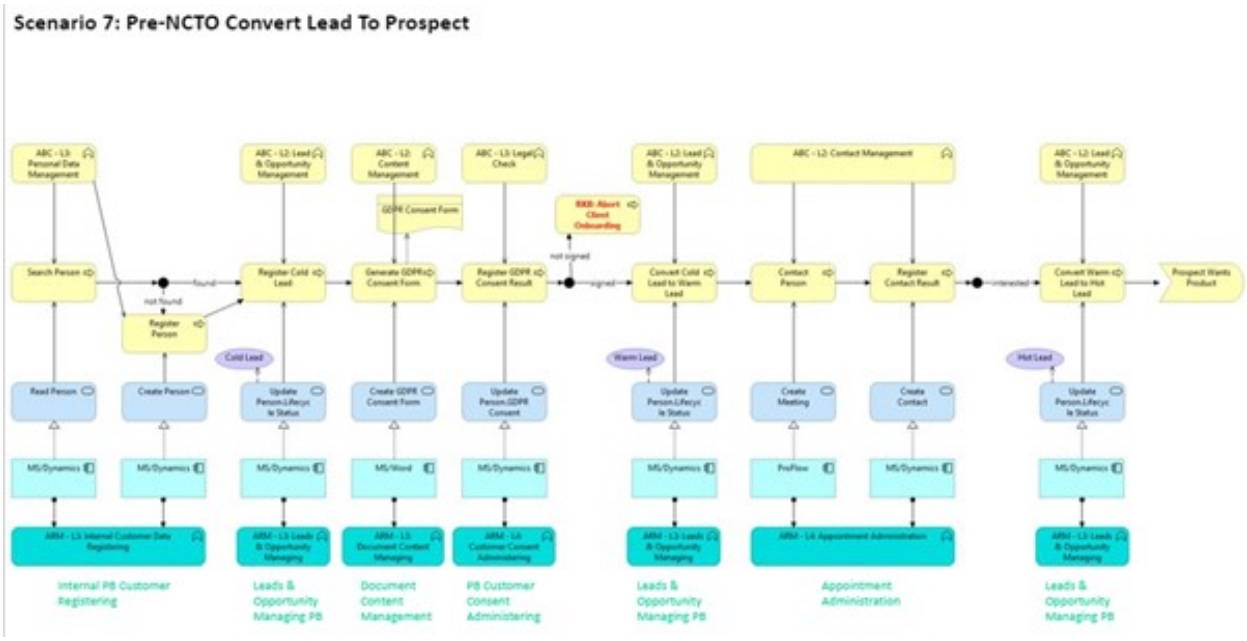


Figure (18) – ArchiMate representation of chosen scenario

Figure (18) shows the resulting ArchiModel representation of the scenario chosen, Convert Lead to prospect. This step was focused on showing how a different data instance is visualized, how it flows in the system, and results in different consequences. This, in turn, made a scenario very clear from the data object triggered by the user to the interface and the ending result. In this case, the main focus was the customer.

As the project advances, the team created a model and identified the model's different data sources Figure (19). The team took a step to identify whether a data source on the model was automated – presented in green – Semi automated – presented in yellow – manual – presented in red – and described the data sources in terms of more information about their goals.

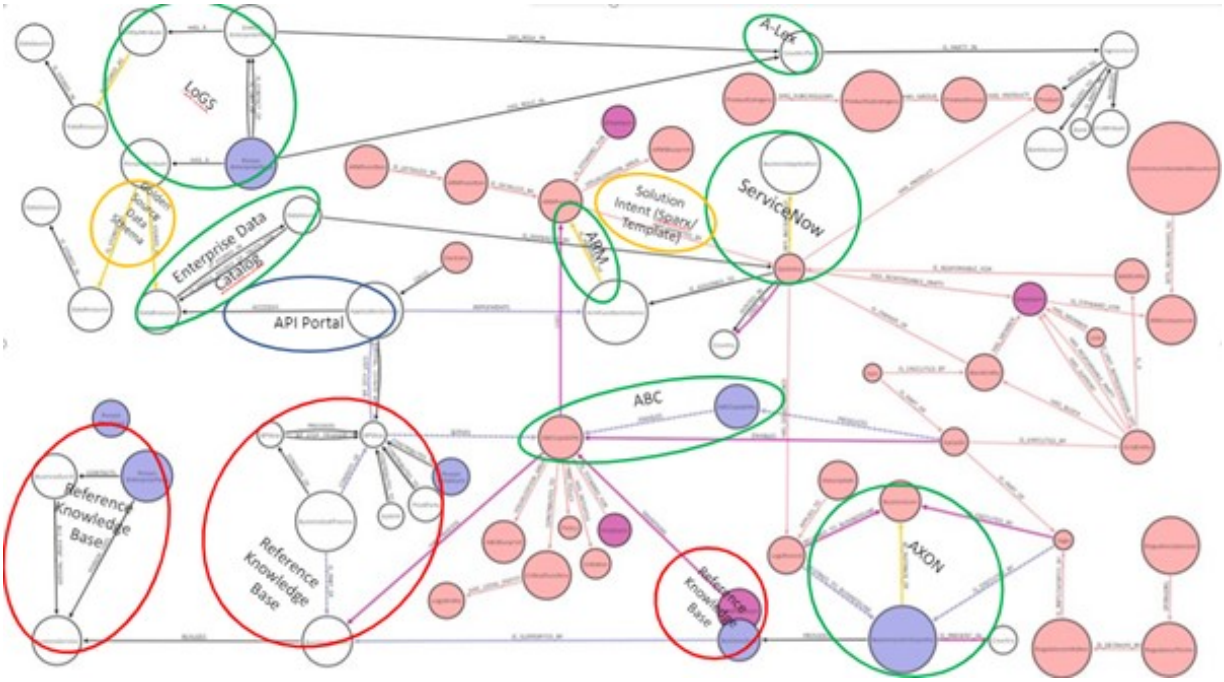


Figure (19) – model with classified data sources

As the case study states, the team's effort was integrated with a different team with goals that overlap with the one in the study. The other team had more focus on the business side than the project owner and the IT's side, which resulted in difficulties in the next steps. However, the primary step was the sixth step, which is where the overlap, differences, and additions were identified.

In this step, the model showcased in the appendix shows a full view of the datasets and the applications with the relationships between them. Here, the team was able to identify all the elements and the relationship and add their properties, for example, application interfaces as API, HTTP, etc., and an application service's role and if the service is a microservice.

The team's mission did not end up here as this is only based on the testing data chosen in the third and fourth steps. Therefore, the team requires adding the rest of the scenarios to the database and creating queries based on them. The queries were not prepared to the moment of this write-up. And therefore, the testing of the queries is missing.

7.4. Findings & Discussion

This thesis study aims at studying how a company in the financial sector is able to understand their system and work their way through to new technology, microservices in this case. This study has presented a theory and cases from research, and a case study focused on finding a method to be applied to reach the end goal. This section will summarize the findings of the research along with illustrating the journey of the bank.

By splitting the goals into three main achievements, each serving to one of the sub-questions of the research study, it is easier to interpret them and have a check with the team about the process. One achievement is gathering information from the stakeholders of the bank and the system and studying the processes. While the second is analyzing the system currently presented and the last is to begin preparing for the plan of the main implementation. The project's main goal is to be able to step a foot forward in the microservices world.

Presenting the steps and concerns that were in the project and generalizing them into a guide that explains the experience of the bank is the main goal here. Furthermore, a discussion about the key points from the theory and the case study will be integrated within the discussion to ensure clear delivery of the results.

Step 1 – Organizational requirements

At the beginning of any project, the main goal is set and the subgoals are defined as time goes forward. A clear set of issues that are considered as the main reason for the project is then listed clearly and discussed with the team. What is the effect of these issues and why should they be fixed? Two critical questions would be answered by all stakeholders interacting with the system.

Firstly, listing these concerns and translating them into questions. The answers are the expected output of the project. This way, the team can include topics such as performance, resource usage, information technology infrastructure, efficiency, and dependencies, etc. Moreover, the team can now have expectations about what the expected output should look like and how will it be achieved.

However, this is not enough clarity for complex projects and the big environment of banks or corporates in general. With a clear mindset pushed by the project manager into the project team, everyone included will be able to perform with the same end goal and in the same direction.

Team management is a very important aspect of a project with such capacity and timeframe; therefore, the team should be diverse, including stakeholders from different departments and ready to change. Engineers are of no doubt part of the core along with the business developers. The team is needed to work coherently and persistently since companies have shown failure cases as shown earlier in the study because of a lack of motivation of readiness to a new way of developing.

Now that the team is in one boat and ready to move, the questions defined should be used to brainstorm and decide on the included resources. Resources can be study areas, such as processes and departments, employees and stakeholders that affect the process, data that will be considered,

and the tools to be used. Investigating tools that the developers are comfortable with and can be further used as a method to explain to other stakeholders is going to make the process more comfortable.

As the team gathers information and brainstorms, they encounter new aspects that have not been aware of in the beginning and start incorporating them. understanding how other stakeholders deal with data or how they see the process in a different matter could give the developer a lot of insights into the system. Therefore, it is crucial to understand who uses the system, how, and what are the goals of each department from that project.

As this step becomes clear, the team will have figured which datasets and processes will be used to test the environment for the new idea. With this method, the team will have the ability to have a mimicked version of the system to do their studies on future implementations. Doing this makes it more structured and easy to deal with in the case of technologies such as microservices.

Moving forward, it is crucial to have legalities in mind and how processes are monitored to match regulations such as the GDPR. In the discussion process, this has been one of the topics mentioned and there was an emphasis on matching the compliances needed. Therefore, matching the technical and legal aspects of the project is of no doubt beneficial for future steps.

Step 2 – Understanding the current state

Now that some aspects are clear to the team, an iterative process over the recently mentioned questions and goals should start. This process will have the team narrow down their focus instead of gathering ideas and information to starting to investigate examples of processes and datasets mentioned earlier. Moreover, this could increase the chance of asking the right questions from the engineer's side and start visualizing the correct way to illustrate these processes.

Documentations of the results from iterations have to take place along with the new sub-goals discovered in the previous steps. One simple yet effective idea was using simple drawing tools like a pen and paper to draw the processes discussed and connect them like Figure (17) have illustrated a small scale of the designed model. Since the project is also based on services and their scalability, a service-focused mindset encouraged by the manager and the team members is advised.

At the moment where all this has happened; the team will be able to create a meta-model of the system which creates a set of rules and constructs for the further milestone to start. Figures (14 & 15) in the case study the team created the model visualizing and present it to the stakeholders and project managers to have a clear understanding of the system and flow of the processes.

This marks the second step of the findings, creating a clear overview of the system; process resource, data resource, and business logic. It provides the stakeholders with a better view of what they are dealing with and a coherent understanding of dependencies and boundaries.

Step 3 – Preparation for microservices.

Now that the team can see the metamodel it is time to move forward with the goals of the main project. The previous step is then considered as the fundamentals of starting an infrastructural change in the organization and preparing the team while setting their mindset in the right direction.

Many of the steps in the first part are similar to what is about to happen further in the project. Now that the team has discussed different processes with different employees in the organization, it is time to pick one that is suitable for further analysis and study. The process is to be mapped on the final model prepared for this project.

The model is then prepared based on the knowledge gathered and processes picked. This model will be an open project that will continue as the team goes further and has the goal of a detailed understanding of the process and the services included. In the case study presented, the processes were presented in ArchiMate diagrams, for example Figure (18) presenting a lead to prospect scenario.

The team had a list of few scenarios where they could pick from, the models would help them see the flow in the system and the different dependencies. Moreover, these models present a clear data object triggered by the users to the interface resulting in the end scenario.

Now the team is looking into the model to be developed and make sure to identify the model's different sources included. The model should include the different data sources and how they are connected to the services, along with the level of automation in this process. These should be identified and studied with the same goal of the first part of the project; understanding the current status.

After the model has been developed and as the increasing effort of detailing it goes on, the model should then include attributes that serve the end goal. For example, service name, type, role, and for the goal of applying microservices, a Boolean to define the service as a standalone microservice or not. Figure (20 & 21 & 22) are an example of what an outcome of the model would look like.

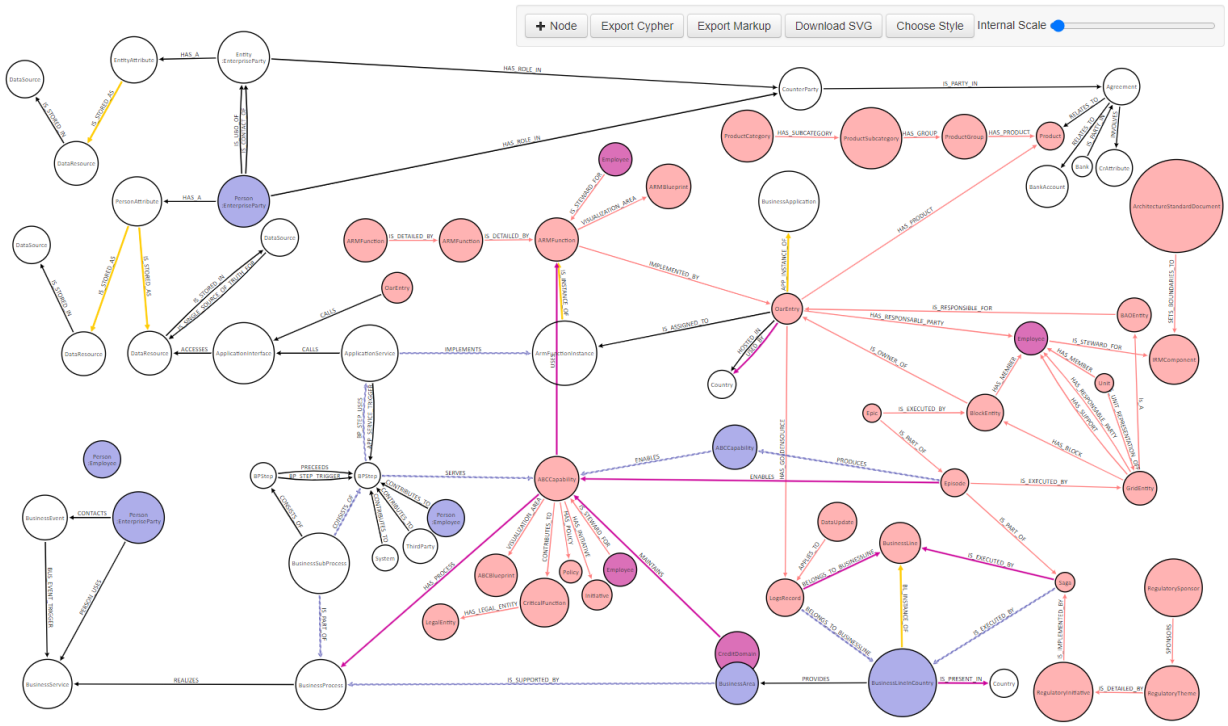


Figure (20) – final model representing the system

The model is also expected to give a new perspective for the team by following the processes and how effective and efficient they are. Therefore, creating a way for them to interpret and suggest alternatives and other routes for the scenarios later on as well as the current state of the architecture. Documentation of these notes is important as they can affect the decision-making in the architectural view of the organization.

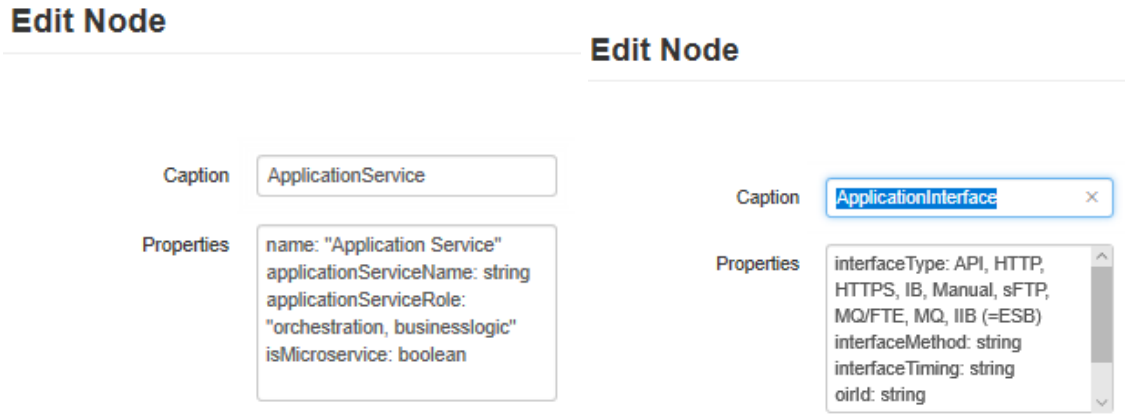


Figure (21 & 22) – showing properties of the nodes

Now putting things to test with the available resources and prepared models. Mapping the processes into the model and discovering the potential of the new technology is in order. Checking how far is the team from answering the questions from the first steps and how this model satisfies

the goal of the project. Going around for a round of discussion with managers and stakeholders is recommended.

The team should dig deeper into the transformation and how it should be structured. From the resources found and documented in the literature review, deciding on details such as the preferred or doable size of services and their scope is a good start. Furthermore, the team is still working on a specified scale of a scenario, it should not include more to avoid complexity and confusion.

One important fact to be aware of is that this point could result in not choosing a new infrastructure, rather digging deeper into the provided alternatives discovered in the process. Since microservices have not been favorited by all the cases studied for this thesis, this step could be a checkpoint to further decide on the following steps.

This checkpoint will benefit the team and the whole organization by understanding the best-case scenario for the stakeholders and the final goal. In other words, is the organization ready for a full change or are they better off improving the current state for the time and look further into microservices later?

Furthermore, if the team's choice is to move forward with the new technology, the previous steps are then reusable and could be redone for any process or scenario that would be considered. In other words, the team cannot work on architecture without understanding its different aspects and dependencies. These steps set a clear ground for the following work projects and technical questions.

Ideally, if the answers pointed out to start integrating microservices, the team needs put into considerations the environment, the system's scope, whether the services will be broken down or replaces, and how they are connected as included in the containers and virtual machines sections. Nevertheless, data sources are part of the equation, and therefore how are data connected to the services and achieve efficiency in that matter is in order. Here comes the importance of understanding legal considerations and dependencies such as data protection and retrieval for the customer's information.

In the end, the team will be able to identify the benefits of the technology when it comes to availability, scalability, and deployment for each service. As explained in the related work section, each infrastructure has its characteristics and benefits, and this is where such analysis is put to use. The main goal of a project with this scope has to be upgrading and not downgrading the experience.

The company now has a reusable method to discover and discuss new potentials in their systems. By weighing out the benefits and consequences from the process, they can identify what is best for the company and the affected parties. Moreover, the team is now more aware of what can change and how and have a documented process to go back to when things are unclear.

It is important to take into consideration that this research is based on a case study that is specific to the bank and the results are prone to interpretation and subjectivity of the matter.

Summary of the process

To summarize this section, figure (23) represents a 5 steps approach to the solution presented in this thesis. The process starts with a definition of the goal and the problem in hand, gathering information from the team and stakeholders around and shaping these problems into answerable questions. Secondly, the initiation of the project comes by deciding on which elements of the system will be included and which will be significant to the result of the project. These two steps help to define the basic organizational preparation for the project and the culture promoted in the team.

Furthermore, the next step is to combine the accumulated knowledge and getting into more depth of understanding these elements by using the resources available such as a process model. By combining these resources together with the information gathered, the team can create a metamodel serving the understanding of the current situation of the system. The fourth step is then to study the model and identify points in the system which can be resolved and altered accordingly. In this phase of the project, the team has a clear vision of the current state and what could be done in that regard.

The last step is to translate all that knowledge into a useful model representing the current situation and potential alternatives. These alternatives will then include general improvements in the system to be able to move forward or suffice to solve the current problems. With this being said, this model will also show a basic ground that will help investigate the potential implementation of microservices in the cases where the team sees relevant and useful.

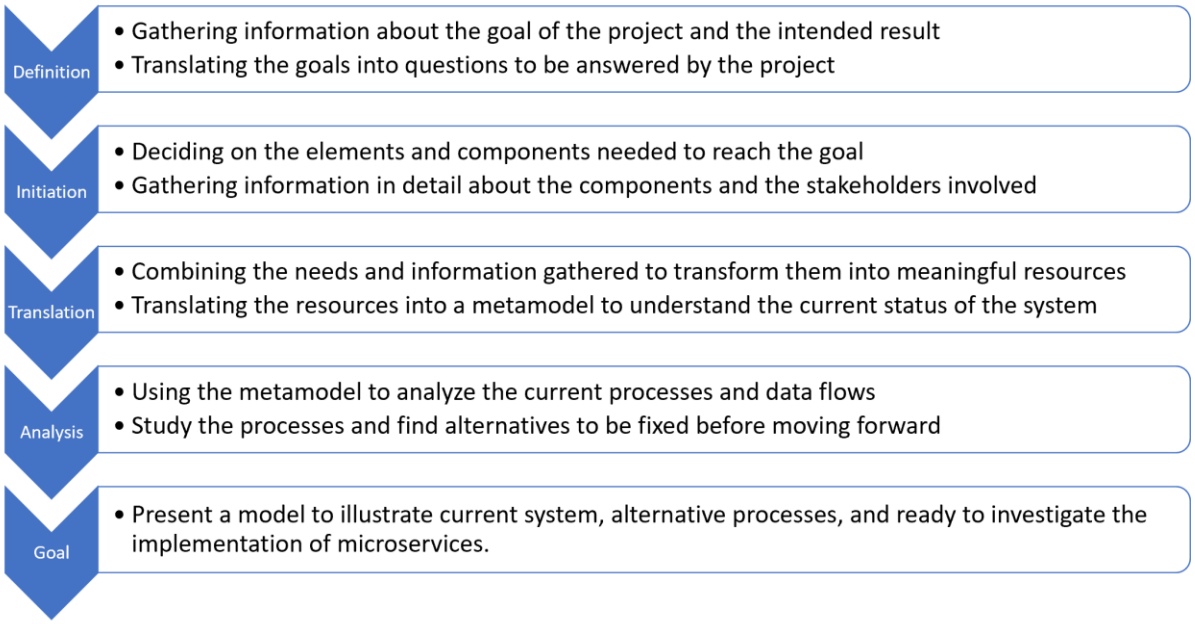


Figure (23) summary of the process resulting in the findings

8. Limitations & recommendations

As the research has been going on, some limitations showed up that made the research restrained in certain areas. In this section, the limitations are mentioned and discussed in light of how they affected the results and findings and how they may be avoided for future reference.

To start, the internship that took place for this research has been drastically affected by the COVID-19 pandemic starting about the same time. This situation has led to losing a massive amount of time in field research and discussions with colleagues from the bank. Furthermore, the pandemic majorly affected the communication between the researcher and the supervisors of the research, which made reaching out harder and took more time than it usually would. This being said, the internship period felt like it was not long enough and that more research was needed.

Moreover, the literature provided on the topic of microservices was somehow limited, for the topic is relatively new. In order to retain the credibility of the research, resources used were restricted to academic peer-reviewed documents and case study write-ups by professionals in the field.

For further studies, a more extended period for the research, along with the presence of a dedicated team, is recommended. This recommendation would ensure having a unified goal, mentality, and focus. Guidance for this project is crucial, along with the motivation done by the project owner. In an agile environment, similar projects could benefit from multiple iterations and discussion of the progress.

The decision of such transformation should be considered after weighing the challenges to be faced after understanding the system. Moreover, as the study of microservices is still not mature like other studies in the innovation world, more questions would arise with time and need more research.

One way to move forward is to identify dependencies in the system, useful quality codes, and take it from there to start questioning the implementation's best start.

9. Conclusion

In this thesis, a study has been conducted on the transformation of monolithic architecture to microservices in different fields. When it comes to monolithic architecture, it was declared that it is working, and companies have gotten used to it. However, with the research, this architecture has shown slower development cycles to maintain effective systems or companies. Continuous development has also been proven to be more problematic when the deployment process gets longer and harder. This only means that reacting to a new trending technology and implementing it in the system would be expected but challenging.

Furthermore, an explanation of microservices was presented in terms of being several small containers, and each includes a service that communicates together. This architecture is seen as a potential solution for the problem of scalability and expandability of the software. These containers can be introduced in different languages and focused on performing their task as one piece of service. Therefore, the services are then able to be deployed, developed, and scaled separately from the system as a whole.

The small-sized containers can help development teams implementing new features to their systems without worrying about the downtime or the consequences on the rest of the system. Nevertheless, customer experience can be enhanced as fixing bugs and editing the code of any of the services no longer takes time with this technology. This way, the continuous deployment pipelines and the feedback received on the new services will enable the business to make faster and more reliable decisions regarding features and developments.

Therefore, microservices can introduce room for more innovation than a company might be used to with their traditional architecture. However, microservices architecture does not come for free. There exist multiple organizational and technical challenges in the implementation of migration to reach the maximum level of benefits from the microservices architecture.

This thesis study's question was how to understand these challenges and whether to transform from monolithic to microservices architecture. Whether the challenges can be overcome, or the organization is not ready yet for this transformation. Moreover, are the already existing challenges in the current monolith solvable on their own or not.

In order to answer this question, a case study has been done with the Dutch bank ABN AMRO along with the research study of this thesis. For 9 months, the case study took place in Amsterdam in the IT innovation department and consisted of studying the current system. Many questions have been asked, followed by modeling the processes and systematically studying them.

The models have shown that there existed a few alternatives that could be presented in the current way the processes are happening. Furthermore, with the focus of the project on the architecture, the team worked on this project with achieving transparency of the system in their mind. As the project goes, not only the IT department had that aim, but it turned to be a bank-wide focus is achieving clarity.

This project has served the bank in further understanding their system. They went in steps to start the project and resulted in a more straightforward path into decision making. This transparency of the processes will then help the stakeholders and executives be more confident when taking a big step towards the transformation.

The first step of the project is to ask the questions that will help the decision-making process and finding a way to answer them. By using the team's expertise, a model has been designed connecting departments, processes, and data. Testing this model on some of the processes and sourcing the data into the model was then the next step. This step was a milestone where a presentation was done to executives to prove the project's viability. Querying this model would be a way to answer the questions picked at the beginning of the project.

With microservices in mind, the project helped in understanding the system better. The project might not conclude a tough decision to transform the implementation of microservices on a specific date but aided the team point at a model where microservices would be implemented as a start and opened a way for discussions.

The study has, however, shown that technical capabilities are not the only requirement for the transformation. Working on this project requires a lot of focus and an open mentality for change and variations. They require not only coherent work but a lot of communication and guidance from experts. This internship along with the literature has shown potential in some companies but also a failure in some, thus concluding the need for more research and cases to prove its maximum potential.

The internship duration was not enough to see the final result of the project but showed the beneficial impact of studying the system and its relationships. The transformation to microservices is a long journey, but that should not stop any company from moving forward and providing its best. This transformation can give a bank, in this case, a competitive edge by implementing all their newer features faster and with less waiting time.

Finally, the steps presented in this study could be reused by any development team to make better decisions on their systems and understand further the impact of their processes. These steps can either help a company in its decision for microservices or just improve their efficiency through their current monolith.

Bibliography

Al Debagy, O., & Martinek, P. (2018). A Comparative Review of Microservices and Monolithic Architectures. In *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)* (pp. 149-154). Budapest: IEEE.

Balalaie, A., Heydarnoori, A., Jamshidi, P., Tamburri, D. A., & Lynn, T. (2018). Microservices migration patterns. *Softw Pract Exper, 48*(11), 2019-2042. Retrieved from https://doi.org/10.1002/spe.2608

Barashkov, A. (2018, December 4). *Microservices vs. Monolith Architecture*. Retrieved from Dev: https://dev.to/alex_barashkov/microservices-vs-monolith-architecture-4l1m

Bauer, R. (2018, June 28). *What's the Diff: VMs vs Containers*. Retrieved from BackBlaze: https://www.backblaze.com/blog/vm-vs-containers/

Bill Doerrfeld. (n.d.). *From monolith to microservices: Horror stories and best practices*. Retrieved from TechBeacon: https://techbeacon.com/app-dev-testing/monolith-microservices-horror-stories-best-practices

Boner, J. (2016). *Reactive Microservices architecture.* O'Reilly Media, Inc.

Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S., & Mazzara, M. (2018). From Monolithic to Microservices: An Experience Report from the Banking Domain. *IEEE Software*, 50-55.

Campanella, F., Della Peruta, M. R., & Del Giudice, M. (2017). The effects of technological innovation on the banking sector. *Journal of Knowledge Economy*, 356-368.

Daya, S., Duy, N. V., Eati, K., Ferreira, C. M., Glozic, D., Gucer, V., . . . Vennam, R. (2015). Transforming a monolithic application to use microservices (CloudTrader). In *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach.* IBM Redbooks.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow. *Present and Ulterior Software Engineering*, 195-216.

Flygare, R., & Holmqvist, A. (2017). *Performance Characteristsics between monolithic and microservice-based systems.* Karlskrona: Belkinge Institue of Technology.

Fritzsch, J., Bogner, J., Zimmermann, A., & Wagner, S. (2019). From Monolith to Microservices: A Classification of Refactoring Approaches. *Lecture Notes in Computer Science*, 128-141.

Gnatyk, R. (2018, October 3). *Microservices vs Monolith: which architecture is the best choice for your business?* Retrieved from N-ix: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/#:~:text=Better%20scalability.,is%20no%20need%20in%20it.

Gonzalez, D., RV, R., & Sharma, S. (2017). A Solution Approach. In D. Gonzalez, R. RV, & S. Sharma, *Microservices: Building Scalable Software.* Packt Publishing.

I. Mitic. (2020, April 30). *Everything you need to know about online banking: statistics and facts*. Retrieved from Fortunly: https://fortunly.com/statistics/online-mobile-banking-statistics/

Jhangiani, R. S., Price, P. C., Chiang, I.-C. A., Leighton, D. C., & Cuttler, C. (2017). Observational Research. In *Research Method in Psychology.*

Juniper Reasearch. (2020, March 3). *Digital banking users to exceed 3.6 billion globally by 2024, as digital only banks catalyse market*. Retrieved from Juniper Research: https://www.juniperresearch.com/press/press-releases/digital-banking-users-to-exceed-3-6-billion#:~:text=Banks%20Catalyse%20Market-,Digital%20Banking%20Users%20to%20Exceed%203.6%20Billion%20Globally%20by%202024,2020%3B%20a%2054%25%20increase.

Kappagantula, S. (2018, Febuary 27). *Microservice Architecture — Explore UBER's Microservice Architecture*. Retrieved from Edureka: https://medium.com/edureka/microservice-architecture-5e7f056b90f1

Kasireddy, P. (2016, March 4). *A Beginner-friendly introduction to containers, VMs and Docker*. Retrieved from freecodecamp: freecodecamp.org/news/A-Beginner-friendly-introduction-to-containers-VMs-and-Docker-79a9e3e119b/

Kharenko, A. (2015, October 9). *Monolithic vs. Microservices Architecture*. Retrieved from Microservices Practitioner Articles: https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59

Kratzke, N. (2018). *A Brief History of Cloud Application Architectures From Deployment Monoliths via Microservices to Serverless Architectures.* Preprints.

Maimbo, H., & Prevan, G. (n.d.). *Designing a Case Study Protocol for Application in IS research* . Perth: School of Information Systems, Curtin University.

Mark Boyd. (2016, August 8). *How to succeed at failure with microservices*. Retrieved from TheNewStack: thenewstack.io/succeeed-failure-microservices/

Mat'sela, M. (2019, May 13). *Microservices Architecture vs. Service Oriented Architecture*. Retrieved from Mualle Mat'sela: https://mualle.net/microservices-architecture-vs-service-oriented-architecture/

Mazzara, M., Dragoni, N., Nucchiarone, A., Giaretta, A., Larsen, S. T., & Dustdar, S. (2018). Microservices: Migration of a Mission Critical System. *IEEE Transactions on Services Computing*.

Muhammad Raza. (2019, October 9). *System Reliability and Availability Calculations*. Retrieved from bmc: https://www.bmc.com/blogs/system-reliability-availability-calculations/

Nehme, A., Mahbub, K., Jesus, V., & Abdallah, A. (2019). Securing Microservices. *IT Professional*, 42-49.

Nielsen, C. D. (2015). *Investigate availability and maintainability within a microservice architecture.* Aarhus: Aarhus University.

Observational Research. (2019, June 19). Retrieved from Provalis research: https://provalisresearch.com/blog/observational-research/#:~:text=The%20term%20refers%20to%20the,are%20neither%20controlled%20nor%20manipulated.

Prasad, R. (2019, August 16). *Application Scalability — How To Do Efficient Scaling*. Retrieved from DZone: https://dzone.com/articles/application-scalability-how-to-do-efficient-scalin

Rubens, P. (2017, June 27). *What are containers and why do you need them*. Retrieved from CIO: cio.com/article/2924995/What-are-containers-and-why-do-you-need-them.html

Rudrabhatla, C. K. (2018). Comparison of Event Choreography and orchestration Techniques in Microservice Architecture. *International Journal of Advanced Computer Science and Applications*, 18-22.

Salkind, N. (2010). Encyclopedia of Research Design. *Thousand Oaks, CA*. doi:10.4135/9781412961288

Sarsansig, A., & Leon, F. T. (2019). Performance analysis of microservices and microservice architecture - conatainers technology: proceeding of the 7th international conference on software process imporvement (CIMPS 2018). In *Trends and applications in Software Engineering* (pp. 270-279).

Sharma, S., RV, R., & Gonzalez, D. (2017). *Microservices: Building Scalable Software.* Packt Publishing.

Singhal, N., & Chelliah, P. R. (2019, January). Selection Mechanism of Micro-services Orchestration Vs. Choreography. *International Journal of Web & Semantic Technology, 10*, 1-13. doi:10.5121/ijwest/2019.10101

Stelligent. (n.d.). *How one global bank moved from Monolithic to efficient through automation on AWS.* Mphasis Stelligent.

Stenberg, J. (2014, August 14). *Experiences from failing Microservices*. Retrieved from InfoQ: https://www.infoq.com/news/2014/08/failing-microservices/

Steven Lemon. (2019, August 9). *Why our team cancelled our move to microservices*. Retrieved from Medium: https://medium.com/@steven.lemon182/why-our-team-cancelled-our-move-to-microservices-8fd87898d952

Stubbs, J., Moreira, W., & Dooley, R. (2015). Distributed Systems of Microservices Using Docker and Serfnode. *7th Interntaional Workshop on Science Gateway*, 34-39.

Tapia, F., Mora, M. A., Fuertes, W., Aules, H., Flores, E., & Toulkeridis, T. (2020, August 21). From Monolithic Systems to Microservices: A comparative study of Performance. *Applied Sciences*. Retrieved from https://doi.org/10.3390/app10175797

The Reactive Manifesto. (2014, September 16). *The Reactive Manifesto*. Retrieved from The Reactive Manifesto: reactivemanifesto.org

Villamizar, M., Castro, H., Merino, V. M., & Casallas, R. (2015). *Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud.* Colombia: ResearchGate.

Wix Engineering. (2015, July 14). *The Microservices and DevOps Journey at Wix*. Retrieved from Wix Engineering: https://www.wix.engineering/post/the-microservices-and-devops-journey-at-wix

Yadav, A. (2018, Jan 10). *Why companies like Netflix, Uber, and Amazon are moving towards Microservices*. Retrieved from Techsur: https://techsur.solutions/why-companies-like-netflix-uber-and-amazon-are-moving-towards-microservices/

Yoav Abrahami. (n.d.). *Scaling Wix to 60M users - From monolithic to microservices*. Retrieved from StackShare: https://stackshare.io/wix/scaling-wix-to-60m-users-from-monolith-to-microservices

Appendix

Transparency in IT PB&CIB

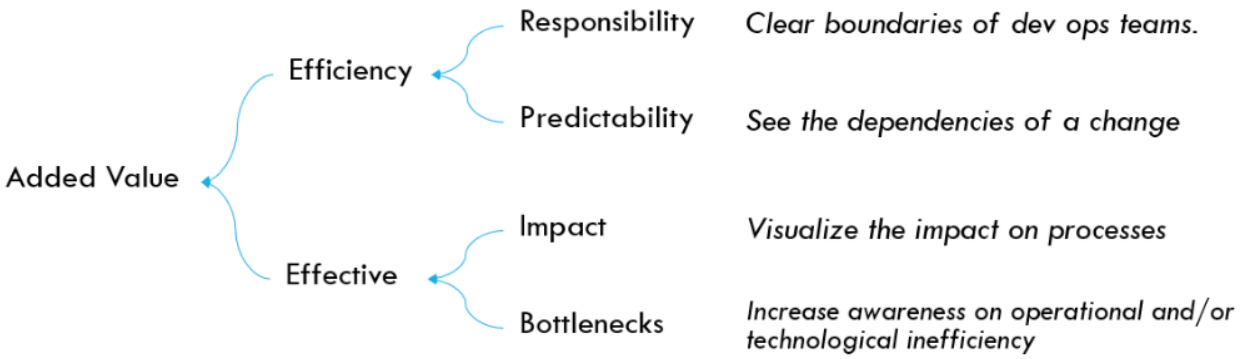
Mission

Transparency in IT PB&CIB

Linking applications to process and technology.

Creating a source for tactical and operational decision making.

By modelling and showing the relations between Banking processes and Information Technology, the organization will have more transparency on the added value they provide.



Final Goal

Goal for our team:

A Neo4J graph Database that holds different components and can show their relations :

PB&CIB Banking processes: (pre-)NCTO process (first data to import)

Information Technology: incorporate the elements that are in Clarity (ABC, ARM, IRM)

Be able to query this database and visualize the answers.

Goal of the Clarity team:

Clarity is a governance tool for (business) architecture, to support decision making by management. The goal for clarity is to get insight into the various IT developments, in relation to the business capabilities.

Insights expected are:

- Where does the bank spend its money on change?
- What are the strategic key capabilities for the bank?

- Which grid is busy with what capability?
- What is the impact of legislation and regulations?
- Where are we using 'Software as a Service'? Etc.

Clarity provides insight by analysing data from 'execution' with data from 'architectural models'. Execution data comes from costs related to Applications, Grids, and Blocks. They are compared with architectural models ABN AMRO Business Capability (ABC), Application Reference Model (ARM) and Infrastructure Reference Model (IRM), etc.

The tool is also monitoring the mapping of applications to these models, looking into trends.

Goal in Collaboration with Clarity:

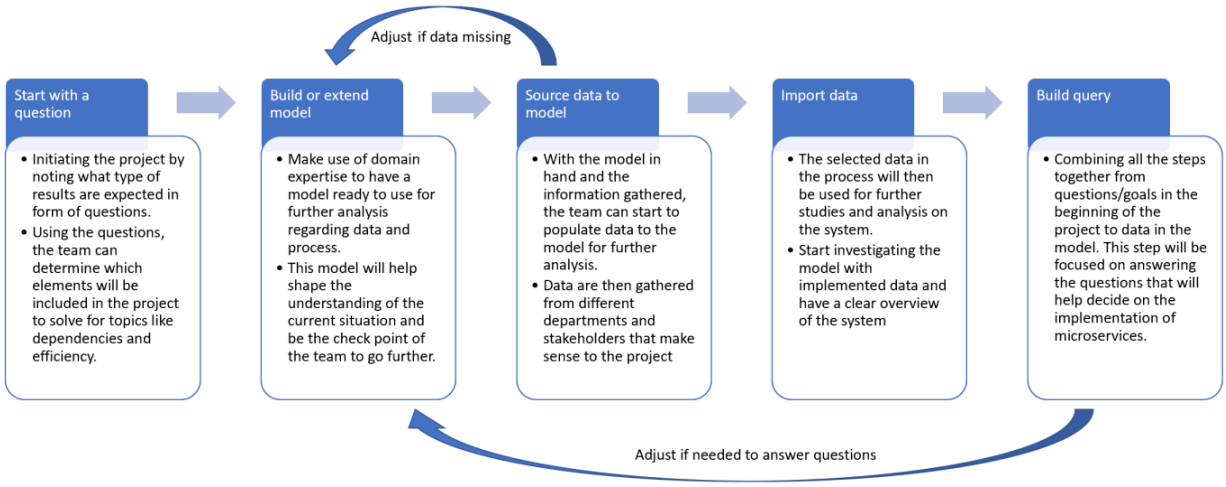
By incorporating the elements from clarity, we can merge our database with Clarity's database by:

Adding our data (IT International) to Clarity

Adding our resource model elements (nodes and relationships) that are not yet present in Clarity

Proposing changes to the Clarity database model where we have/need alternative properties/labels/nodes/relationships.

Main approach



1. Start with a question

Which questions need to be answered by querying the database?

- Make clear about the responsibilities
- Predictability, which parts of a system are affected by change?
- Impact - is the technology having a malfunction?
    - which processes are affected and how are they otherwise?
- Multiple servers for data attributes? how many places does a piece of data exits?
    - how many versions of the data do we have?
- How are we calling elements?
    - different data to different database? mapping into the elements? *

Recently added questions

- Where and how many times do we store the name of a Client?
- What is the single source of truth for the name of a Client? Are there more than 1 ?
- Can the address of a Client be changed in a 'non-single source of truth'?
- What BusinessLines in which countries is an Application used by?
- What processes are affected by application Service X being down?
- What options are available for an Application Service to retrieve a Client's address?

Identify which CSA (Current State Architecture) **elements need to be represented in the model.**

- Application Component
- Application Service
- Application Interface
- ARM function
- BusinessProcess
- BusinessArea
- BusinessLine
- EnterpriseParty
- Contractual Relation
- DataSource

2. Build a model, linking applications to process and technology

Create a Meta-Model (structure of essential elements) to map data components onto.

Use the advantages of a Graph Database (like Neo4J)

- schema-optional
- using multiple labels to group elements
- variable properties (not all elements with the same label need to have (all of) the same properties

Set up the model in a way that will enable queries to ask the questions from Goal 1.

How to model a (neo4j) graph database

- What modelling choices are there to be made?
- What examples of modelling It architecture, linking applications to process and technology?

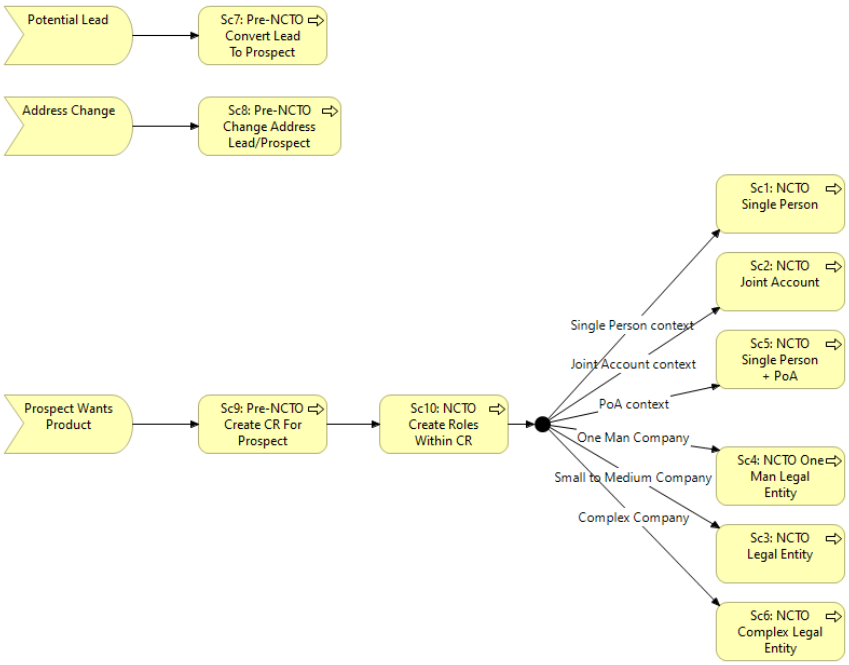
<u>Build a model, using domain expertise (ongoing process)</u>

<u>Model, visualized</u>

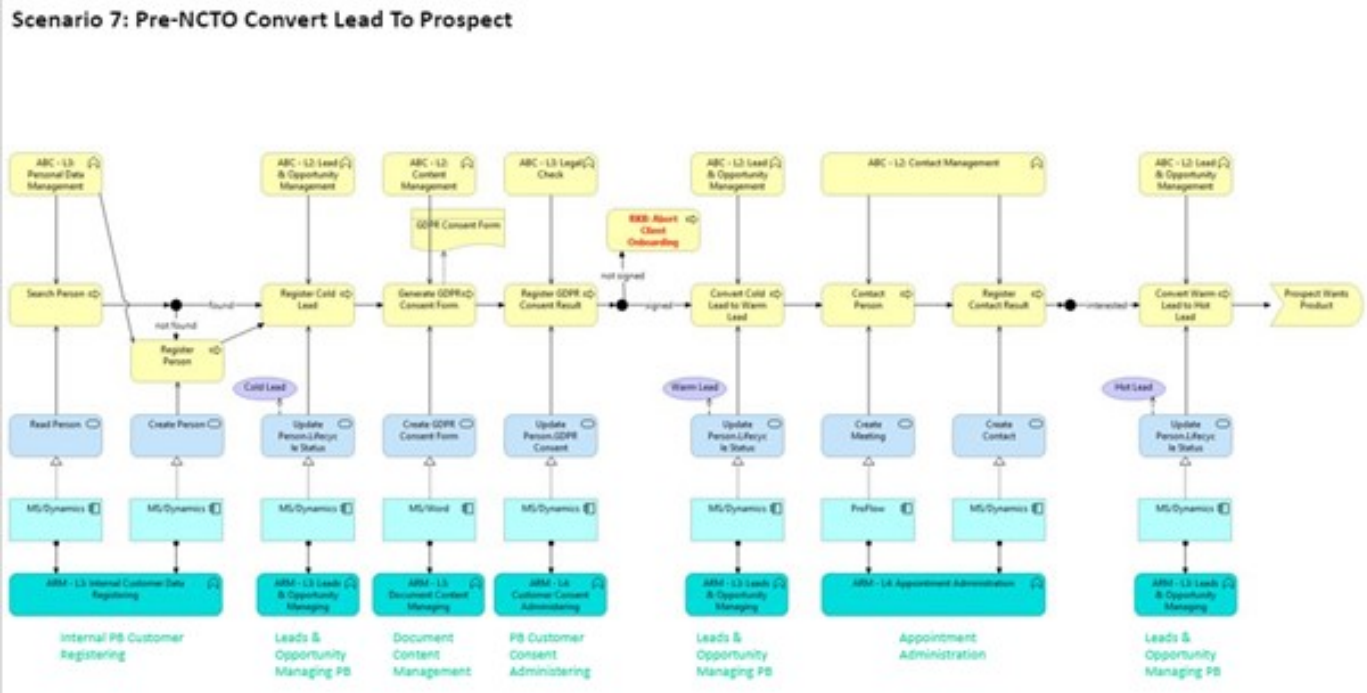
3. Map a process (NCTO) onto the Model

Map a process (NCTO) onto the Meta-model.

Overview of all Scenarios of the NCTO process: Business Process Decomposition NCTO – Archi Models



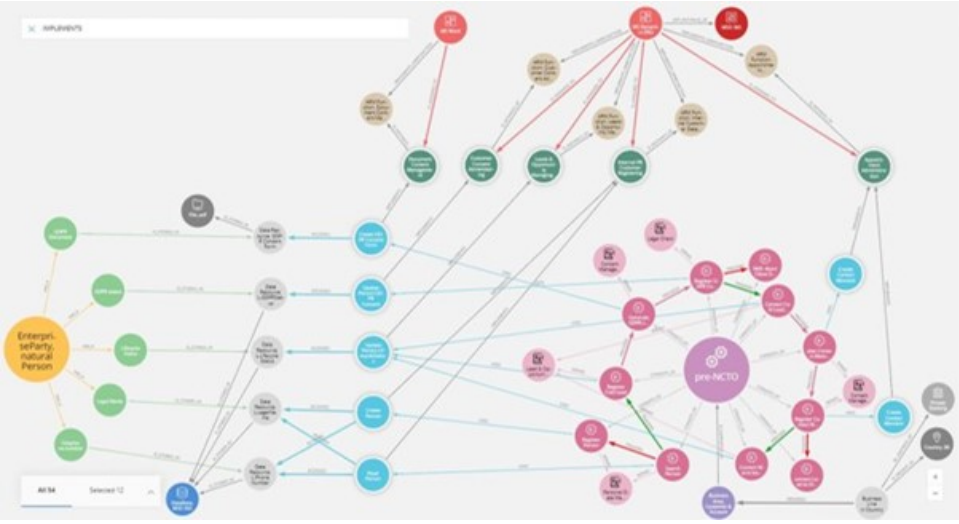
Model, containing scenario7 (pre-NCTO process)



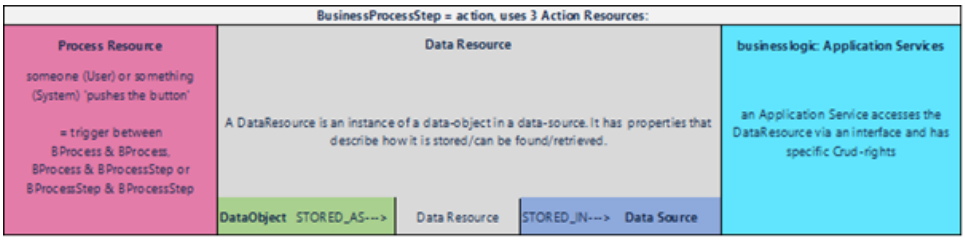
4. Create a Neo4j Database of the model, containing all components relevant to scenario7.

Neo4j database of scenario7

Presentation



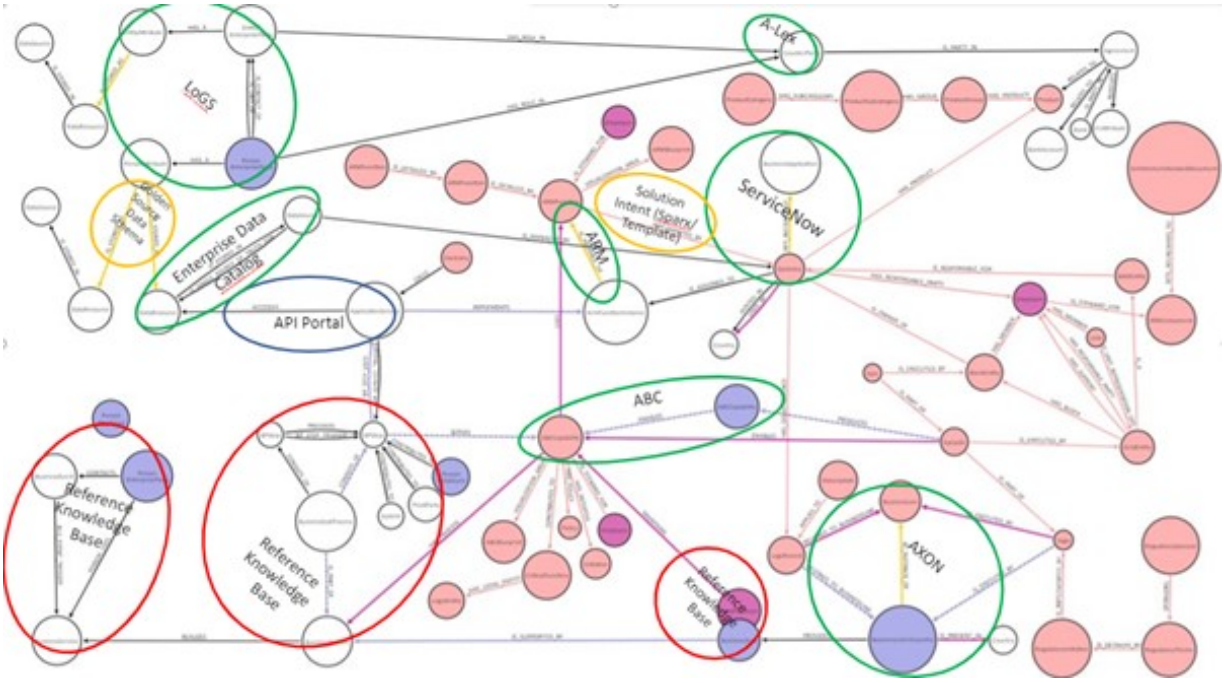
Visualize the Neo4J Database to see the steps in the process.



Screenshots of Bloom for presentation.

5. Identify the Data Sources

Repository Mapping



Green – Fully Automated

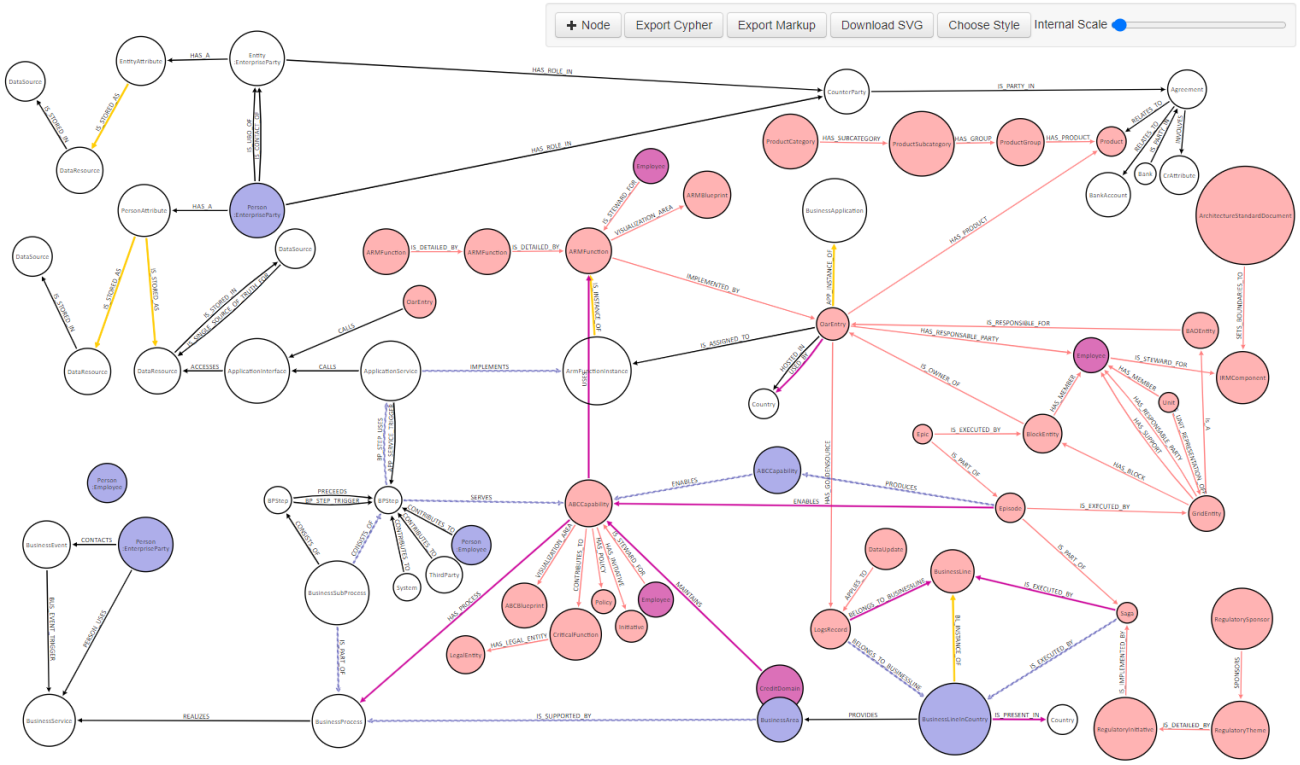
Yellow – Semi automated

Red – Manual

Datasets

dataset	Info	goal	source
The Finance & Risk Information Model (FRIM)	the essential source for Finance and Risk business terms and definitions	A credit facility is an agreement between the Bank and a counterparty under which the Bank allows the Counterparty the right of use, up to an agreed maximum, of one or more types of financing products. The credit facility potentially leads to a claim on that Counterparty or, in the case of debt securities, a financial obligation by a debt security issuer to the Bank in the latter's capacity of investor in a public debt issue.	: ABN AMRO Lexicon (already available in Clarity)
ABN AMRO Business Capability Model (ABC Model)	combine resources, competencies, information, processes, and technologies	deliver their specific value proposition.	Already available in Clarity
Application Reference Model (ARM)	is a classification and description of Application Functions. ARM represents the automated behaviours (application functions) the information systems possess.	ARM answers a simple question: 'What functionalities current or planned applications of the Bank perform?'	Already available in Clarity including relationship with ABC
The List of Golden Sources (LoGS)	the Authoritative List for all Golden Data Sets, with their Data Owners and Golden Sources.	LoGS allows Data Users to quickly find the Data Owner of a golden data set, as well as the golden source of the data where it is managed and kept accurate.	Already available in Clarity? including relationship with Data elements and systems
List of applications and components			OAR already available in Clarity incl. relationships with ARM
List of interfaces			OIR XLS Model Bank only
CMDB			
Sparx EA:			

6. Map the Clarity model onto our Model to identify overlap, additions, differences.



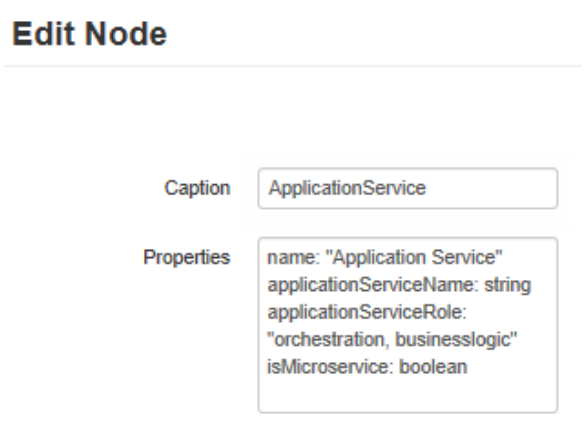
Light Pink: Elements and Relationships that are in the current Clarity Database that overlap or can be used by us too.

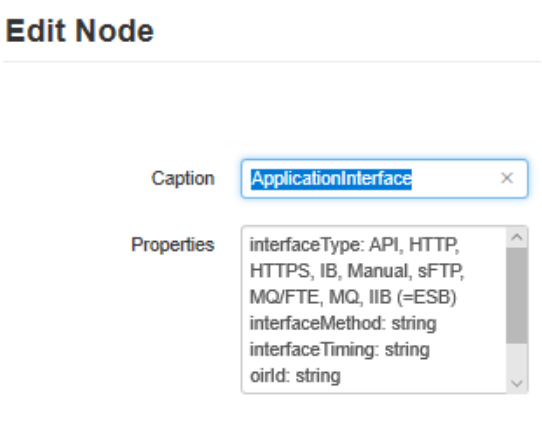
Fuchsia Pink: Elements and Relationships in the current Clarity Database we want to discuss/ suggest an alternative to.

Blue/lilac: alternatives we want to suggest

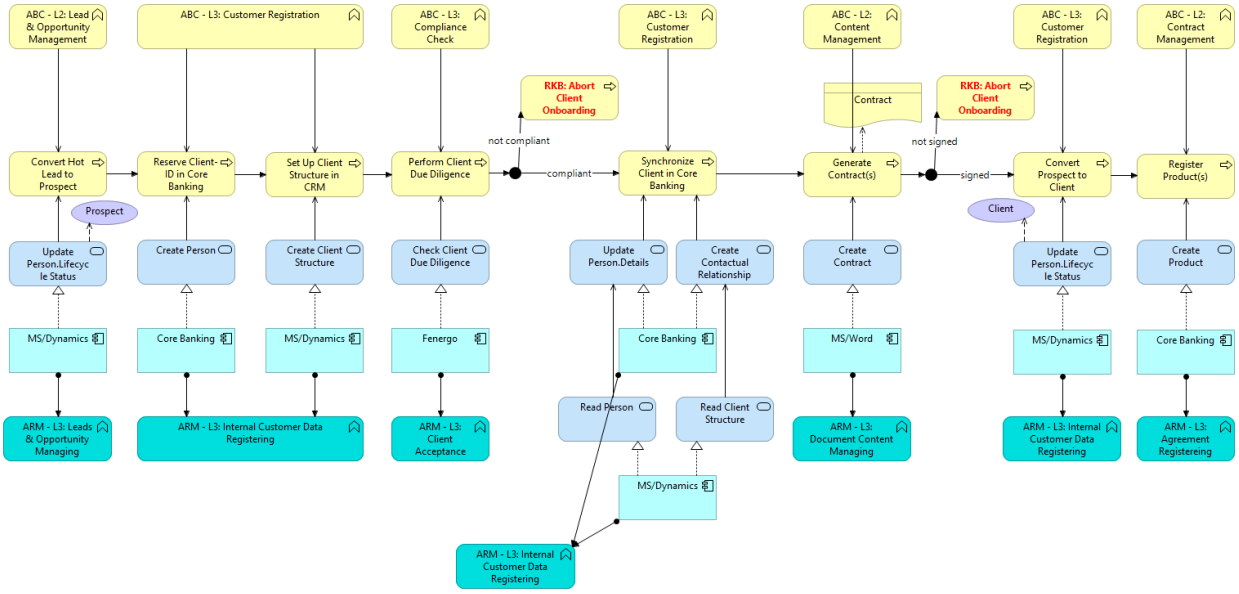
White: Nodes and Relationships we want to add to the Clarity Database.

Yellow: Relationships between elements and their instances.





7. Add NCTO process scenario 1 to the database.



8: Add other Data from Data-sources to the model

9: Query the Neo4J Database to be able to answer the questions.