Measuring the Modeling Complexity of Microservice Choreography and Orchestration: The Case of E-commerce Applications

By

Mahtab Haj Ali

Supervised By

Prof. Morad Benyoucef

Thesis submitted to the University of Ottawa
in partial Fulfillment of the requirements for the
Master of Science in Digital Transformation and Innovation

University of Ottawa

Table of Contents

Abstract

With the increasing popularity of microservices for software application development, businesses are migrating from monolithic approaches towards more scalable and independently deployable applications using microservice architectures. Each microservice is designed to perform one single task. However, these microservices need to be composed together to communicate and deliver complex system functionalities. There are two major approaches to compose microservices, namely choreography and orchestration. Microservice compositions are mainly built around business functionalities, therefore businesses need to choose the right composition style that best serves their business needs.

In this research, we follow a five-step process for conducting a Design Science Research (DSR) methodology to define, develop and evaluate BPMN-based models for microservice compositions. We design a series of BPMN workflows as the artifacts to investigate choreography and orchestration of microservices.

The objective of this research is to compare the complexity of the two leading composition techniques on small, mid-sized, and end-to-end e-commerce scenarios, using complexity metrics from the software engineering and business process literature. More specifically, we use the metrics to assess the complexity of BPMN-based models representing the abovementioned e-commerce scenarios.

An important aspect of our research is the fact that we model, deploy, and run our scenarios to make sure we are assessing the modeling complexity of realistic applications. For that, we rely on Zeebe Modeler and CAMUNDA workflow engine.

Finally, we use the results of our complexity assessment to uncover insights on modeling microservice choreography and orchestration and discuss the impacts of complexity on the modifiability and understandability of the proposed models.

Acknowledgments

I would like to express my sincere gratitude to the following people, without whom I would not have been able to complete this research!

Foremost, I would like to say a very special thank you to my supervisor, Dr. Morad Benyoucef, for his wonderful support and encouragement throughout this research, and especially for his trust and confidence in me. His effective guidance and supervision have made this academic journey an inspiring experience for me. I also thank my colleague, Razib, for his support and collaboration in this research.

Words cannot express my wholehearted gratitude to my parents and beloved sister for their unconditional love and support despite the distance between us. Thanks for giving me the strength to reach for the stars and chase my dreams.

And of course, thank you God, for always being there for me!

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
MSA	Microservice Architecture
SOA	Service Oriented Architecture
REST	Representational State Transfer
MQ	Message Queue
ESB	Enterprise Service Busses
CDM	Canonical Data Model
BP	Business Process
BPMN	Business Process Modeling Notation
CEP	Complex Event Processing
GUI	Graphical User Interface
DSL	Domain Specific Language
XML	Extensible Markup Language
LOC	Lines of Code
DSR	Design Science Research
CFC	Control-Flow Complexity metrics
MCC	McCabe's cyclomatic complexity metrics
DSM	Durfee square metric
PSM	Perfect Square Metric

PC	Procedure Complexity
IC	Interface Complexity
CNC	Coefficient of Network Complexity
CLA	Connectivity Level between Activities
TNA	Total Number of Activities
NSFA	Number of Sequence Flows between Activities
NOA	Number of Activities in a workflow
NOAC	Number of Activities and Control-flow in a workflow
NOAJS	Number of Activities, Joins, and Splits
TNT	Total Number of Tasks
TNCS	Total Number of Collapsed Sub-processes
CoC	Coefficient of Connectivity

1 Chapter 1. Introduction

There has been an ongoing progress in the architecture of software systems over the last few decades, leading to a need for more distributed and modularized systems. Such advancement in software architecture has shifted service-oriented computing towards a more loosely coupled approach using microservices (Mazzara et al., 2017). In a traditional service-oriented architecture (SOA) application, the entire system relied heavily on one single executable artifact that uses one programming language or framework, resulting in more complicated code bases in the system architecture. Therefore, making a change to the system can be challenging as the system grows over time resulting in tightly coupled monolithic services with very little cohesion in coding. This is why fixing and debugging code are complex in monolithic applications (Newman, 2015).This monolithic approach in designing systems brought several limitations to the systems. One of the main drawbacks of such architectural style is that system maintenance is a hard and complex task since a small change in one entity can affect the entire system, therefore resources cannot be allocated efficiently based on the need of each single service. There is also more possibility of a single point failure in the system (Nehme et al., 2019)**.**

However, microservices (a more complete definition will be provided later) are implemented as independently deployable services that can do only one specific business function, leading to less complexity in service implementations. Given the no share standard in microservice architecture (MSA), each service uses its own database, which helps reduce the dependency between services. There is also less chance of single point failure as each microservice operates independently. Hence, failure in one microservice will not affect the entire system. Moreover, since MSA is composed of multiple microservices, it is possible to use different technologies to meet the requirements of each service and avoid choosing one standardized technology, which increases the robustness of the code. Microservices are constantly scaled in the system and communicate based on an event-driven mechanism which is handled by an event store, called event broker (Nehme et al., 2019). So, when an event is called to perform a task, another event might be triggered to help complete the task. Therefore systems that are built on microservices are more productive, cost efficient and scalable, thanks, in part, to the cloud-based architecture of microservices (Baboi et al., 2019).

As mentioned earlier, MSA based applications are composed of multiple microservices that are designed to perform one single task. However, these microservices need to collaborate with each other to complete their tasks and achieve the final outcome in the application. Therefore, it is important to define a communication mechanism between microservices in one application. This mechanism is called service composition and is realized through two main composition methods: choreography and orchestration (Rudrabhatla, 2018a; Singhal et al., 2019b; Valderas, 2020). Some recent studies have also suggested using a combination of choreography and orchestration which is called a hybrid composition method (Singhal et al., 2019a; Valderas, 2020). Obviously, these composition styles have pros and cons. Therefore, companies need to choose the right composition style to fit their software applications' requirements and accomplish their business needs. Yet, this remains a challenging task since every business has different standards and requirements.

The objective of this thesis is to compare the complexity of models (i.e., microservice compositions) built using the two major microservice composition methods. Our research domain is focused on e-commerce applications. There are two main reasons for choosing e-commerce for our research domain. First, given the rapid growth in information and communication technology (ICT), there has been an extensive utilization of e-commerce applications for businesses to compete in the market and grow their revenue and customer satisfaction. According to Asrowardi et al. (2020), e-commerce applications have increased significantly as a way for companies to promote their business and increase their revenue. Secondly, with the global pandemic of COVID-19, which hit the world in December 2019, there has been a huge transformation in the way businesses operate (Bhatti et al., 2020), e-commerce being one of the tools for such transformation. The main focus of our research is on microservices. For that, we conduct a thorough literature review on microservice architecture and microservice composition to identify the characteristics of microservices, and understand the issues surrounding their composition. Armed with such knowledge, we will then design e-commerce applications using the two leading microservice composition styles, run those applications to make sure they work properly, and then assess and compare the complexity of the underlying models.

1.1.Major Underlying Concepts

Service-oriented architecture (SOA), is an architectural framework in which business applications are broken down into several services that communicate via a communication protocol such as Enterprise Service Busses (Xiao et al., 2017).

Microservice Architecture (MSA) is an upgraded version of monolithic SOA that consists of many loosely coupled services that work independently and communicate via messaging and REST APIs (Baboi et al., 2019).

Microservice composition is a mechanism that defines how multiple microservices collaborate in a business application to achieve a system's requirements. Microservices use two different styles: Centralized (orchestration) and decentralized (choreography) to communicate with each other (Bigheti et al., 2019).

Business Process Model and Notation (BPMN) is a high standard language that is used to visualize business processes (BPs). The main purpose of using BPMN diagrams is to give a clear understanding of BPs for various professional roles including business analysts, developers, and stakeholders (Bocciarelli et al., 2012; Valderas et al., 2020). BPMN diagrams are designed at the early stages of the process lifecycle to define more readable business processes which can later be used by professionals to automate and maintain the systems (Bocciarelli et al., 2012).

Business process complexity metrics are numerical expressions of a BP model's complexity and structure. They demonstrate a quantitative measurement to increase the maintainability and ease the process of forecasting errors in business models (Banerjee, 2018). There are various BP complexity metrics inspired by the complexity measurements used in software engineering. In Chapter 5, we will introduce and explain the current state of the art on metrics used for measuring the complexity of BP models. These metrics can specifically help us in measuring the complexity level in microservice orchestration and choreography to understand the models' difficulty level with regards to their understandability and maintainability.

1.2.Research Questions

The goal of our study is to design BPMN-based microservice orchestration and choreography models and assess and compare their complexity. To achieve this goal, first, we define multiple e-commerce scenarios and use the two composition techniques to model, implement, and run those scenarios within a microservice architecture development environment. Then, we use various methods for measuring BPMN complexity to evaluate our models and draw conclusions. Hence, we need to find answers to the following questions:

- **What are the main differences between choreography and orchestration considering the advantages and disadvantages of each composition style?**
- **Which composition technique (orchestration or choreography) is less complex to deliver business requirements in e-commerce applications based on the proposed scenarios?**

1.3.Research Objectives

In order to answer the research questions stated in the previous section, we need to achieve the following list of objectives.

1.3.1. Achieve a good understanding of choreography and orchestration and the differences between them.

In order to understand the differences between these two concepts, we need to conduct a literature review that consists of six steps: I. Defining research questions. II. Choosing the right search strategy by defining search queries. III. Running the initial search to select the primary studies, which we perform in two phases in this research. IV. Evaluating the papers considering their relevance to the topic of our research. V. Establishing a strategy to extract data from the selected papers. VI. Selecting synthesis methods to compare and evaluate the papers.

1.3.2. Classify different scenarios in e-commerce applications.

First, we define what e-commerce is and how it has affected the shopping patterns and habits among users. We then classify multiple scenarios based on different shopping processes on e-commerce websites.

1.3.3. Use Zeebe Modeler as a tool to design choreography and orchestration diagrams using Business Process Model & Notation (BPMN).

Based on the abovementioned scenarios, we model and run e-commerce applications as choreographies and orchestrations using BPMN in a microservice architecture development environment. Zeebe Modeler (Zeebe.io, n.d.)is used as a modeling tool, which features an intuitive BPMN 2.0 modeler that makes it easy to create professional business process diagrams. This tool also features choreography and orchestration modeling frameworks which serve the purpose of this research. Zeebe Modeler also allows us to make executable diagrams using Zeebe-docker, CAMUNDA automation engine and Zeebe Simple Monitor to deploy and execute our models (Zeebe.io, n.d.). The tool generates XML code for each model, which we later use as one of the ways to measure the complexity of our models.

1.3.4. Conduct a thorough study about business process complexity metrics in the literature.

We use complexity metrics to assess and compare the various e-commerce scenarios modeled as microservice orchestrations and choreographies. To that end, we study the literature to find the state of the art on complexity metrics used to assess business processes.

1.3.5. Apply the metrics.

After we identify the existing complexity metrics in the literature, we select the ones that can be adapted to our models. For that, we consider the metrics that use the same BPMN elements as the ones applied on our models to measure the level of complexity in each model.

1.3.6. Draw conclusions.

We use the results from our measurements to compare the level of difficulty for microservice choreography and orchestration and discuss the impacts of complexity on the understandability and modifiability of the models.

1.4.Research Contributions

The study contributes to the field of e-commerce by understanding which microservice composition style (orchestration, choreography) is best for developing e-commerce applications. We achieve this via designing executable BPMN diagrams to implement choreography and orchestration on different e-commerce scenarios and then use complexity metrics to assess and compare the predefined models.

Another contribution of this thesis is academic in nature as there are limited studies which focus on assessing and comparing the two leading microservice composition styles. Thus, this study will shed light on the pros and cons of microservice composition styles based on their complexity.

1.5.Research Motivation

Advances in software development have led to a change in systems architecture. Service-oriented systems are migrating from monolithic architecture to a new approach called microservice architecture to solve the existing challenges in software development paradigms (Mazzara et al., 2017; Newman, 2015). Therefore, many businesses are shifting their technologies towards MSA, which allows more flexibility and scalability in their applications (Bucchiarone et al., 2018; Newman, 2015; Shadija, D., Rezai, M., & Hill, 2017). This approach has gained huge acceptance in many business fields. One of the industries that can extensively benefit from MSA is e-commerce since e-commerce applications require multiple services including, for instance, shipment, payment, and user authentication in their deployment. Service scalability and reusability are significant requirements when building e-commerce applications. Therefore, e-commerce companies need to use microservices to enhance their systems (Asrowardi et al., 2020).

Even though microservice architecture enhances the functionality of systems in many ways, it makes the distribution and communication process more complicated as each microservice is designed to do one specific task and needs to collaborate with other microservices to complete

the business needs (Balalaie et al., 2018), known as microservice choreography and orchestration. These two composition styles offer different standards for the way microservices communicate with each other in one application. However, since every business has different requirements, it is important to choose the right composition style to meet the needs of the business.

Although microservice architecture is gaining popularity in the application development industry, it is still a challenge for businesses to implement the right composition to meet their business needs. In addition, there is limited academic research that compares these composition styles based on the level of complexity of the models. Hence, considering these current knowledge gaps, this thesis is centered on providing a thorough comparison between orchestration and choreography in e-commerce applications using complexity as a metric for evaluation.

1.6.The Organization of Thesis

The organization of this thesis is as follows. The second chapter features a thorough literature review about related concepts such as microservices, service-oriented versus microservices, microservice composition styles, e-commerce, and microservices in developing e-commerce platforms. The third chapter focuses on the Design Science Research methodology, which is the methodology used in this research. In chapter four we describe the design a set of e-commerce workflows using BPMN and divide them into three categories. The fifth chapter features complexity as a metric to evaluate and compare the developed models via different techniques, derived from the literature on measuring complexity in BPMN models. We later use the results from the evaluated models to measure our end-to-end workflows and discuss our findings in the context of the theory proposed by Sánchez-González et al. (2010). The last chapter assesses the results of the evaluation to draw conclusions and propose suggestions for future work in this area.

2 Chapter 2. Literature Review

This chapter addresses our first research objective. We first explain the search process that we use to find papers on the subject matter. After we describe the search process in the literature review, we delve into the literature review to get a good understanding of our research questions which focus on 1. Microservices; 2. Differences between microservice architecture and service-oriented architecture; 3. Different microservice composition styles; 4. Complexity metrics for BPMN models.

2.1.Literature Review

2.1.1. Search Query

In order to find the most relevant studies, we use a two-phase data collection approach to cover the areas mentioned above. First, we select some academic databases including Web of Science, Scopus, ScienceDirect, IEEE Xplore, and Google Scholar. We define the following search queries to implement an automatic search in the selected four databases.

1. **(Microservice* OR "Micro-Service*" OR "Micro Service*")**
   AND
   (Compos* OR orchestrat* OR choreograph*)

2. **(Microservice architecture* OR "Micro-Service architecture*" OR "Micro Service architecture*" OR "MSA")**
   AND
   ("Service-oriented architecture" OR "Service oriented architecture" OR "SOA")

3. **(BPMN AND complexity)**

Then we perform a manual search in journals, conferences, and books to find the papers that could not be found in electronic databases. Further, we study the literature referenced in the representative papers. The initial search resulted in the total number of 700 papers.

2.1.2. Selection of Primary Studies

In the second screening round we apply some fitters, listed below, to evaluate the selected papers considering the metadata: title and abstract. We also choose a number of criteria, listed below, for our selection.

- For the search period we choose papers that were published from 2011, when microservices were first introduced, to present.
- Studies that are written in English.
- Studies that focus on topics that help in answering our research questions.
- Studies that specifically address the two microservice composition styles (orchestration and choreography)

The above conditions led to discarding 630 papers because either their main focus is not on microservice composition or they just mention the search keywords in a general manner.

2.1.3. Selection of Final Papers

After discarding the first set of papers, we propose our second search query to get some papers about the comparison between microservice architecture and service-oriented architecture. Then we evaluate the remaining final papers based on the number of times they use the search keywords in the title and abstract considering our search queries. Finally, we get a total of 40 studies from the four electronic databases and the grey literature. Figure 1. represents the percentages of selected papers according to each electronic library.

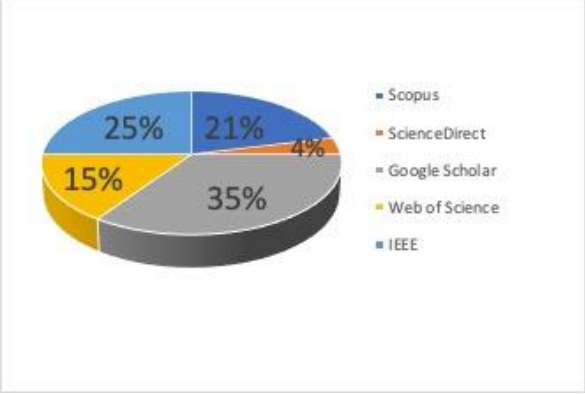


Figure 1. Percentage of primary studies classified by digital libraries

All the final papers focus on the comparison between SOA and MSA and the two methods of composition (choreography, orchestration) in microservice architecture. Figure 2 shows the number of papers based on their year of publication.

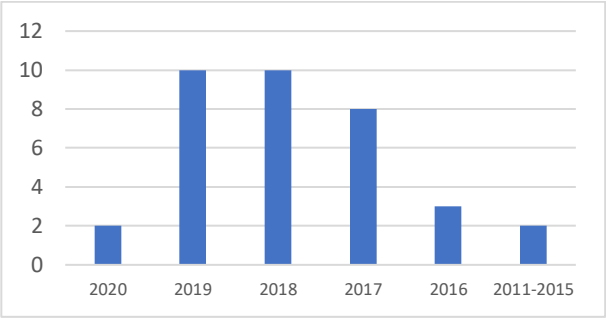


Figure 2 Number of publications classified by year of publication

2.2.What are Microservices?

Microservices are small, autonomous services that collaborate together. The terminology was first discussed at a workshop of Software Architects which took place in Italy in May 2011. Some literature reports define the word “micro” in Microservice as “very small” which means microservice architecture uses many tiny services in one application. However, micro does not always refer to the size of the service since it may differ from one application to another or one coding language to another. Microservices are upgraded versions of SOAs in which applications are composed into smaller services that focus on only one task to increase the scalability and modularity in each software application (Bigheti et al., 2019).

According to Xiao et al., (2017), a microservice is highly scalable and flexible. The main features of a microservice are: each microservice is designed to do a single business task; a microservice can be developed and maintained by a single programmer; multiple microservices can communicate with each other using messages or well-defined APIs; microservices do not share databases; microservices can use shared libraries but do not share source code; each microservice operates independently. Singhal et al. (2019b) also define microservices as self-defined applications that support horizontal scalability and are heterogeneous in nature.

2.3.What is a Microservice Architecture (MSA)?

MSA is considered an upgraded version of a monolithic service-oriented architecture (SOA). The main purpose of MSA is to solve the existing design issues in monolithic and SOA style applications (Baboi et al., 2019). MSA is designed to divide systems into small services that can work independently of each other in one application. Services follow a domain-driven development model also known as bounded context which helps microservices perform tasks with little dependency on other services. These services have their own management pattern and database which is defined as polyglot persistence (Nehme et al., 2019). This feature enables the services to be loosely coupled and use the database that fits their needs. J. Lewis and M. Fowler (2014, p.1) define MSA as "an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies". Microservice architecture can be closely linked with DevOps (Development and Operations) practices and CI/CD (Continuous Integration / Continuous Delivery), since in MSA each microservice can be developed using a different back end and be deployed separately which can result in more complex applications, DevOps practices can help manage the complexity in the application (Baškarada et al., 2018).

There are two methods of communication in microservices: synchronous and asynchronous. In a synchronous communication, a service initiates a request and waits for the response, which blocks the entire application until the task is complete. It works on a request/response basis. However, in an asynchronous approach, there is no central controller to send and receive requests, every service knows what and when to react and instead of asking the server for doing a task it actually does the task and triggers an event so that other services can collaborate and complete the task.

All services communicate via messages using a message broker. Microservices use lightweight mechanisms to transfer messages over hypertext transfer protocol (HTTP). They use the RESTful architectural style as a means of communication. The architecture of a microservice is designed to structure applications as loosely coupled microservices that collaborate with each

other. This architectural style can use various frameworks or protocols since there are no predefined rules, however the protocol has to be lightweight.

While in traditional SOA, orchestration is preferred when composing services; in a microservice architecture there is more focus on choreography. In choreography, services communicate using an event-driven architecture which means every service knows when and how to react either by listening to events on its environment or by using a point-to-point exchange style (Nkomo & Coetzee, 2019).

Rudrabhatla (2018b) expresses a microservice architecture based application as a distributed system in which each service has its own database to act independently from other services.

Mazzara et al. (2017) define MSA as a cohesive process in which services are only responsible to do one single task well. These microservices are mainly business oriented which implement business requirements as services (Mazzara et al., 2017).

Microservices use Representational State Transfer (REST) or Message Queue (MQ) as the interaction methods between services (Shadija, D., Rezai, M., & Hill, 2017).

Viennot et al. (2015) argue that one of the challenges of microservices is data replication as every service uses its own data model, it is necessary to replicate data across the data stores, however data replication in real time is still a big challenge in MSA (Viennot et al., 2015).

2.4.Differences between Monolithic SOA and MSA

MSA is a broader version of Service Oriented architecture (SOA) with a focus on specific features including decentralized data management among services, lightweight services and continuous delivery (Di Francesco et al., 2017). According to Richards, M (2015), one of the main differences is that MSA is designed with the focus of share-nothing philosophy to promote agile features and make services more autonomous, while SOA supports a share-as-much-as-you-can methodology to stimulate a great level of reuse. Another difference is that while SOA relies on both orchestration and choreography, MSA mainly focuses on choreography in composing services (Di Francesco et al., 2017).

In general, there are several factors that distinguish SOA and MSA as shown in **Table 1**. In MSA, services are fine-grained, which means the components of the system are smaller, while in SOA, services are usually coarse-grained which consist of fewer and larger services. Having a

fine-grained component relatively increases the design complexity but can reduce the number of calls among services to complete a task. Considering service deployments, SOA uses heavy middleware orchestration style to compose its services while microservices use RESTful protocols (Shadija, D., Rezai, M., & Hill, 2017).

Both SOA and MSA compose services that are available over a network and integrate across heterogeneous platforms. In SOA the business process is deployed on top of the services which allows the system to make any changes to processes, however, this feature causes dependency by connecting all services to one central context. Whereas, in MSA, each service is only responsible to perform one particular task. While this feature in microservice deployment might lead to some duplicities, it considerably reduces the dependencies across microservices. The other benefit of microservices is that it is fairly easy to scale up the system. This increases the elasticity, scalability in microservice compositions which makes these services more cloud-friendly compared to SOA. Microservices are designed as containers or individual processes. Every service has its own database, but they usually share the same database schema. Microservices do not have a central element to control the interactions across the services therefore they prefer choreography over orchestration. So microservices are responsible for interacting with other services in the system. This might reduce the flexibility in designing business processes over company IT requirements but increase independency among microservices. However, this does not mean we cannot use orchestration in microservice composition. Given the independence of microservices, business processes support a more continuous and scalable service delivery which reduces the chances of failure in the entire system (Cerny et al., 2018).

One other factor to consider when we compare MSA to SOA is the communication mechanism that they use. In SOA, services communicate and function with a high level of dependency using tools such as Enterprise Service Busses (ESB). ESB is a mechanism that is used to distribute work among connected services of an application. It is a centralized tool through which smaller services are routed and integrated to implement business rules. On the contrary, in an MSA, there are end points that follow the business rules. By decentralizing the application architecture, microservices can keep up with any changes in the size of the application and developers are capable of writing more logic inside the microservices (Cerny et al., 2018).

Table 1. Comparison of Monolithic SOA and MSA

Factors	MSA	SOA	References
Scalability	Elastic	Limited elasticity	(Cerny et al., 2018; Singhal et al., 2019b; Systems et al., n.d.; Thramboulidis et al., 2018; Xiao et al., 2017; Yahia et al., 2016)
Database	Per microservice	Shared	(Bogner et al., 2019; Cerny et al., 2018; Rudrabhatla, 2018a; Systems et al., n.d.; Viennot et al., 2015; Xiao et al., 2017; Zimmermann, 2016)
Cloud-native	Yes	No	(Cerny et al., 2018; Mazzara et al., 2017; Ortin & O'Shea, 2018; Singhal et al., 2019b; Thramboulidis et al., 2018; Zimmermann, 2016)
Deployment	Independent	Dependent/Monolithic	(Cerny et al., 2018; Singhal et al., 2019b; Thramboulidis et al., 2018; Xiao et al., 2017; Yahia et al., 2016; Zimmermann, 2016)
Service size	Fine-grained/small	Coarse-grained	(Cerny et al., 2018; Singhal et al., 2019b; Xiao et al., 2017; Zimmermann, 2016)
Service communication	Choreography/ Orchestration	Orchestration	(Baškarada et al., 2018; Camilli et al., 2018; Monteiro et al., 2018; Oberhauser & Stigler, 2017; Rudrabhatla, 2018a; Scattone & Braghetto, 2019; Singhal et al., 2019b, 2019a; Thramboulidis et al., 2018; Valderas, 2020)
Management	Distributed/Centralized	Centralized	(Cerny et al., 2018; Xiao et al., 2017)

Message routing	API gateways, lightweight messaging	Enterprise Service Buss (ESB)	(Cerny et al., 2018; Zimmermann, 2016)
Service architecture	Independent, uncoupled services running in their container	Loosely coupled, cohesive	(Bogner et al., 2019; Xiao et al., 2017)
Deployment	Independent	Dependent/Monolithic	(Cerny et al., 2018; Singhal et al., 2019b; Thramboulidis et al., 2018; Xiao et al., 2017; Yahia et al., 2016; Zimmermann, 2016)

2.5.Microservice composition styles

The purpose of microservice implementation is to design systems into small, loosely coupled and independent services. However, the main challenge is how to coordinate these services to meet the needs of large business applications (Monteiro et al., 2018). In other words, composition defines how different services communicate with each other in a business application to solve complex business needs. Microservice composition targets the mapping of microservice units and functionalities to develop the architecture which is lightweight and independently deployable. There are two ways to compose microservices: centralized (orchestration) and decentralized (choreography) (Bigheti et al., 2019; Singhal et al., 2019a). Orchestration is a common technique in SOA which can also be used in MSA. However, microservices are mainly focused on decoupling services and eliminating dependencies among services which is why more focus is on choreography as a dominant style to compose and design services (Cerny et al., 2018; Di Francesco et al., 2017; Nkomo & Coetzee, 2019; Valderas, 2020). Although choreography is an ideal choice for MSA, some scholars argue that implementing choreography increases the complexity in the applications which makes it very challenging for developers to manage the entire application without a central controller in place (Rudrabhatla, 2018a). Below we will talk about each composition style briefly.

2.5.1. Orchestration

This composition style provides accurate and trusted interactions between services. In orchestration, all service interactions are controlled by a central controller that functions similar to an orchestrator (Rudrabhatla, 2018a; Singhal et al., 2019b). The orchestrator is responsible for the entire communication in the system. All the requests and service calls are managed by the central controller. This centralized environment uses request/response messages as a communication mechanism. In this composition approach, the central controller calls a service by sending a request to that service and waits for it to respond before sending a request to the next service (Monteiro et al., 2018; Rudrabhatla, 2018a; Singhal et al., 2019b). The next service cannot be called until the called service sends the proper response to the orchestrator. This increases the waiting time and dependency between services (Valderas, 2020). In other words, if one service is down, the other services will never be called. These service dependencies increase the risk for single point failures in the application given the fact that the orchestrator is the only controller and coordinator in the entire system. This can result in the failure of the entire application (Singhal et al., 2019b).

However, this composition pattern makes microservice applications more manageable and less complex to coordinate. Considering the drawbacks of this pattern, it is important for developers to figure out where to place the orchestration in the business applications to get the best outcome (Baškarada et al., 2018; Nehme et al., 2019). It is ideal to execute orchestration at the gateway level. The gateway is an entry point to the system, which is how clients, as external elements, communicate with the system. The gateway receives requests from a variety of clients and thanks to its central position, it can also customize responses based on the client needs and type (Nehme et al., 2019). The gateway later sends the requests to microservices to process the requested tasks.

2.5.2. Choreography

Microservice architecture is designed to reduce dependency on services in one application. In choreography, there is no central controller, so services work independently. The output of a service is the input of another service. Services communicate through messages or patterns with well-defined APIs, run in separate processes and have their own databases (Rudrabhatla, 2018a;

Singhal et al., 2019b). Every microservice can be designed, located and updated independently with no dependency on other services. This approach uses an event-driven architecture pattern for microservices which makes this approach relatively complex compared to the orchestration pattern (Baškarada et al., 2018; Isoyama et al., 2012; Rudrabhatla, 2018a). Reactive architecture patterns can be applied to microservices to reduce complexity in choreography (Monteiro et al., 2018; Singhal et al., 2019b). In a reactive architecture, services are designed as a set of events, which use an event stream for asynchronous communication. In other words, services know what and how to react without a central controller in place. One event can be used by several services and every service can create their own events and send them back into the event stream. Thanks to the asynchronous feature of choreography, there is no waiting time for one service to respond while in orchestration every service has to wait for the orchestrator to be called (Singhal et al., 2019b). In this technique, each microservice performs its own task and communicates with other services to complete complex tasks and get the right result.

According to Peltz (2003, p.46) "Choreography allows each involved party to describe its part in the interaction. Choreography tracks the message sequences among multiple parties and sources rather than a specific business process that a single party executes" (Peltz, 2003, p.46).

2.5.3. Hybrid Composition

Microservices integrate with each other using a centralized (Orchestration) and decentralized (Choreography) composition styles. Applications that use lighter solutions and communicate in an event-based fashion are developed based on choreography. However, orchestration offers a less complex composition style compared to choreography and can be used to analyze the entire composition by using a process model. Therefore, both of these approaches have their pros and cons and some papers have suggested using a clever combination of both techniques, also known as hybrid, based on the needs and the logistics of the business to improve performance (Valderas, 2020). One way to use a hybrid approach is to compose the entire system based on a choreography, which means microservices use an event-based choreography to cooperate with each other. However, orchestration can be used within a specific microservice, for instance if one specific microservice needs additional services, it uses orchestration to communicate with its additional services and then integrates with the rest of the system using a choreography style.

This way the application benefits from the decentralized nature of choreography, while keeping the deployment process less complex using centralized controllers for specific services (Singhal et al., 2019a; Valderas, 2020).

2.6.A comparison between choreography and orchestration

Microservice composition shows service collaborations, the business process and the sequence of the activities in one application. In other words, it is mainly used to deploy and coordinate services in a business application. In this section, we use the literature to evaluate the advantages and disadvantages of microservice choreography and orchestration as shown in **Table 2**.

In an event-based choreography style, when a microservice performs a transaction, it creates an event that can be used by other microservices in the application to initiate their local transactions. This process continues until all the services publish their events. In other words, this process ends when there are no more events to be broadcasted (Baškarada et al., 2018; Rudrabhatla, 2018a).

The other composition style is orchestration in which there is a coordinator that listens to the events published by any of the microservice's local transactions and assigns the next task (transaction) to the right microservice based on the incoming event. The performance of an event-based choreography is quicker than orchestration. Therefore, it is a suitable option for applications with limited number of microservice calls where time is an important element. While timing is relatively higher in orchestration method, this technique reduces the complexity of error tracking in the system considerably thanks to the presence of a central orchestrator at a single location (Rudrabhatla, 2018a).

Service choreography is only responsible for defining rules for how services should interact and exchange messages with one another and the control element is not restricted to a single location (Cerny et al., 2018). This feature increases the level of scalability in choreography-based systems. For example, if we want to add a service in an orchestration-based system, the controller needs to be changed to be able to work with the new service. However, with a choreography approach it is easy to add a new service since services listen to the events and react

when they are triggered. But one of the challenges with choreography is that there is no central service (conductor) to trace the tasks and make sure that all the required actions are completed successfully (Butzin et al., 2016). In order to solve this issue, Butzin et al. (2016) suggest implementing an additional service that only monitors microservices that are performing a task not triggering the events. This way we can keep track of successful tasks and have the flexibility of adding new microservices to the application.

Cerny et al. (2018) also describe orchestration as a canonical data model (CDM) in which several system elements agree and use a standardized data model for the exchange of business objects. In most cases, the whole system ends up with only one business object, which makes future changes to the system a big challenge since all the elements of the system have to agree on the new change. This model also increases the dependencies between services. As a solution to this limitation, Cerny et al. (2018) define a new approach called "bounded context". In this approach every service is only responsible for one single business task in a specific context. Therefore, there is no need to consider all services for every business process. For instance, to process Shipments, only the service responsible for the user's address is called. This way we divide a large model into smaller parts, which also enables the systems to model business objects based on their needs. In other words, services can function completely independently based on their needs (Cerny et al., 2018).

Nevertheless, there are still debates on the pros and cons of both composition styles. On the one hand, several studies argue that choreography is preferred in MSA thanks to multiple reasons (Di Francesco et al., 2017; Nkomo & Coetzee, 2019; Valderas, 2020). The first advantage of choreography over orchestration is that it composes more loosely coupled services that are flexible and easily amendable. However, orchestrated applications have high maintenance cost as it is not easy to apply changes to the system over time (Singhal et al., 2019b). In addition, the communication method in orchestration is on a request/response basis which causes a lot of latency and dependency among the services. As Newman (2015, p.247) also mentions: "Avoid approaches like enterprise service bus or orchestration systems, which can lead to centralization of business logic and dumb services. Instead, prefer choreography over orchestration and dumb

middleware, with smart endpoints to ensure that you keep associated logic and data within service boundaries, helping keep things cohesive."

On the other hand, some studies suggest that orchestration is more doable for developers to implement in their systems as it makes it less complex to develop and manage systems using microservice architecture (Rudrabhatla, 2018a; Singhal et al., 2019b). According to Valderas et al. (2020), decentralization, which is the main focus of microservices, seems to be more achievable in choreography. However, in choreography the overall logic flow is disseminated among different services in the application, there is no big picture of the entire composition which makes it difficult to visualize and understand how the system works. It also makes future maintenance and updates more difficult, i.e., when engineering decisions are needed to add a new service or make changes to the existing microservices. Since there is no big picture, tracking errors is also complicated and time consuming. Furthermore, when microservices are choreographed, they are forced to learn the coordination requirements to communicate with other microservices in addition to performing business tasks that are assigned to them which increases the level of complexity in the system (Valderas et al., 2020).

To benefit from the advantages of both approaches, there are studies that suggest a combination of these architectural styles in a single application (Hybrid method) based on the needs and the logistics of the business (Singhal et al., 2019a; Valderas, 2020).

Table 2. Choreography Vs. Orchestration

Features	Orchestration	Choreography	Authors
Security	✓ High level of security due to the central controller	✓ Low security monitoring due to the lack of central controller	(Singhal et al., 2019b)
Localization	✓ Better choice for local dimensions	✓ Better approach for global scopes	(Singhal et al., 2019a)
	✓ Easier to achieve	✓ Higher maintainability	(Singhal et al., 2019b)

Maintainability	maintainability in big applications	for small and less complex applications	
Monitoring	✓ Easier thanks to the central conductor	✓ No monitoring as microservices are responsible for monitor their performance	(Butzin et al., 2016; Singhal et al., 2019a)
Error fixing	✓ Easier to detect errors as all the tasks are constantly monitored by the orchestrator	✓ Hard to detect the errors but they can't affect the entire system	(Scattone & Braghetto, 2019; Singhal et al., 2019b)
Scalability	✓ Offers low scalability as it's hard to add a new service	✓ Offers high level of scalability since all services work independently and a new service can be added more easily	(Butzin et al., 2016; Singhal et al., 2019b)
Speed	✓ More latency due to send/request communication technique	✓ No or very little latency due to event-based communication	(Rudrabhatla, 2018a; Singhal et al., 2019b; Valderas, 2020)
Complexity	✓ Less complex and easier to manage as there is a central controller to assign tasks and handle the communication in the entire system	✓ More complex since a developer in charge of one microservice has no access to what's happening (i.e., the inner workings) in other microservices	(Baškarada et al., 2018; Isoyama et al., 2012; Rudrabhatla, 2018a; Shadija, D., Rezai, M., & Hill, 2017; Valderas, 2020)

Dependency	✓ More coupling	✓ No coupling or loosely coupling	(Cerny et al., 2018; Hasselbring & Steinacker, 2017; Nkomo & Coetzee, 2019; Singhal et al., 2019b; Valderas, 2020)

2.7.State of the art of microservice composition approaches

In this section we discuss the different technologies and approaches for microservice composition found in the literature. New approaches are proposed to improve microservice composition. For example, Monteiro et al. (2018) introduce an approach called Beethoven, which supports an event-driven platform to compose microservice orchestration using data flows. This platform consists of an orchestration language, which is a Domain Specific Language (DSL) that focuses on workflow, task and event handler, and a reference architecture, which has been represented using the Spring Cloud Netflix ecosystem. However, according to Valderas et al. (2020), the only issue with Beethoven is that developers need to learn a new DSL or use proprietary tools.

Kouchaksaraei et al. (2018), also suggest a framework called Pishahang, which manages and orchestrates cloud-based microservices. The proposed framework integrates SONATA (an orchestration-service framework) with a multi-cloud tool (Kouchaksaraei et al., 2018).

Indrasiri and Siriwardena (2018), introduced Ballerina which is built as a programming language that is used to integrate microservice orchestrations. Their approach uses the programming level to represent the communication between microservices.

Gutiérrez–Fernández et al. (2017) have used BPMN engines to integrate microservice orchestrations. Their approach only focuses on orchestrating microservices using a central controller (Gutiérrez–Fernández et al., 2017).

Valderas et al. (2020), also propose an approach that uses BPMN models to create a microservice composition. Their study focuses on defining an event-based choreography composition style that allows developers to compose decoupled microservices with a possibility of providing a clear picture of the workflow in the system via a centralized mechanism. They develop a microservice composition that allows companies to benefit from both choreography and orchestration. To that end, first, they use BPMN orchestration to propose a clear picture of the workflow among microservices. Then, they split their model into multiple fragments. Each fragment consists of one single microservice using an event-based choreography. In their model, each microservice executes its BPMN fragment independently and creates the event which triggers the next microservice in the process to execute the next fragment (Valderas et al., 2020).

In light of the above, we can argue that all the papers that introduce a framework to compose microservices, such as Pishahang, Beethoven, and Ballerina provide a solution for microservice communication at the programming level, while Gutiérrez–Fernández et al. (2017) and Valderas et al. (2020) have used BPMN modeling to represent microservice communications at a higher level which explains how microservices communicate in a more elaborative fashion. Hence, developers don't need to learn a new technique or a Domain Specific Language (DSL) to use their framework, which also increases the expressiveness of their design. That is why, in this research, we propose BPMN modeling to visualize microservice compositions of e-commerce applications.

2.8.Microservices in E-commerce Applications

2.8.1. What is e-commerce?

Electronic commerce or e-commerce has affected businesses and users' shopping habits in many ways. Businesses can access a wide range of consumers both locally and globally and users can also get immediate access to any types of products with no time or geographical restrictions. E-commerce websites have multiple layers and components, which play an important role in the success of online business. These components include: electronic catalog, secure web server, e-commerce engine, database management and graphical user interface (GUI) (Safavi, 2009).

Usability is an important feature in designing an e-commerce website. Therefore, the design of an e-commerce website plays an important role in the shopping process of users (Belanger et al., 2006). Users make their purchase decisions based on the quality and usability of the website (López-Miguens & Vázquez, 2017). There are several factors including design, the variety of products, product presentations, and usability that impact the user shopping experience making them more involved in their shopping activity. The main elements of an e-commerce graphical user interface are customer registration, products, search and browse, user's feedback and comments, product recommender, product descriptions, customer support, checkout process, and delivery and shipment (Baumeister, 2019; Chu et al., 2007).

2.8.2. How MSA affects e-commerce applications

In this section we explain the role of MSA in implementing e-commerce applications. Hasselbring & Steinacker (2017) explain the use of microservices in one of the biggest European e-commerce platforms called otto.de. They discuss how microservices improve scalability, reliability and agility of the otto.de website by using a vertical structure. They define a vertical domain as a "shared nothing" standard since the verticals are stateless and do not share a database, cache or any other resources and communicate through REST APIs. This will reassure that failure in one service will not impact the entire system with a snowball effect. The other feature of a microservice architecture is loose coupling and high consistency, which is achieved by proposing a transaction-less communication between microservices to keep the data consistent across the system with little dependency among services on an e-commerce platform. However, this approach is not possible in a monolithic application as they use transactions to maintain consistency which causes considerable coupling in the system. One other important function of microservice composition is fault tolerance and resiliency of the system. Given the cross-functional design of microservices, services work dependently and failing of one service will not affect the entire system (Hasselbring & Steinacker, 2017).

The focus of our study is to perform a model complexity-based comparison between the leading microservice composition styles (choreography and orchestration) in developing e-commerce applications.

In particular, we propose a manifesto of microservice composition styles by:

1. Designing BPMN models to give a clear representation of how microservices communicate in both choreography and orchestration.
2. Using the designed models to execute high-level e-commerce scenarios following MSA protocols to build single tasked microservices with a high degree of decoupling.
3. Performing an analysis on the models to measure their complexity and use the results to illustrate how complexity affects the understandability and maintainability of the models.

3 Chapter 3. Methodology

This chapter discusses the step-by-step research methodology that we use to design our models and evaluate the outcome following the **Design Science Research** (DSR) methodology.

3.1.Definition of Design Science Research

The design-science paradigm is basically designed to suggest solutions to existing problems by using various scientific methods to analyze the structure of a system and its real-life applications (Shrestha & Vuorimaa, 2019). This methodology is a common approach in the field of information systems (Peffers et al., 2007). We use DSR to enhance the production, presentation, and evaluation of design science research. Baskerville et al. (2009) define DSR as a paradigm rather than a separate research methodology.

Hevner et al. (2004) propose a seven-step guideline to provide a clear understanding of how to conduct design science research. This guideline helps authors and editors to apply DSR into their research.

The first guideline is “Design as an Artifact” which builds artifacts based on specific problems. The second guideline is called “Problem Relevance” which explains the main purpose of DSR on creating suitable solutions to business problems. The third guideline, “Design Evaluation”, uses evaluation methods to measure the artifacts designed in the first step. The fourth step is called “Research Contributions” which explains how well the designed artifact meets the requirements of the area of research. The fifth guideline, “Research Rigor”, observes the consistency of the artifacts. The sixth step, “Design as a Search Process”, states a process that includes artifact design and search for resources to use the artifact. The last guideline is called “Communication of Research” and is focused on maintaining the communication between both technical and managerial audiences. **Table 3** shows the seven-step guideline suggested by Hevner et al. (2004).

Table 3. Seven Design Science Research Guidelines (Hevner et al., 2004)

Guideline	Description
Design as an Artifact	Design science research develops artifacts in various forms including: a construct, a model, a method, or an instantiation.
Problem Relevance	Design science research develops technology-based solutions to existing business problems.
Design Evaluation	The effectiveness of a designed artifact must be rigorously measured using well-executed evaluation methods.
Research Contributions	Design science research contributes to the areas of the designed artifact, foundations, and methodologies in an effective manner.
Research Rigor	Design science research follows rigorous methods to build and evaluate designed artifacts.
Design as a Search Process	An effective artifact is designed to achieve desired results yet still follows the laws defined in the problem environment.
Communication of Research	Design science research targets both developers and non-technical audiences.

3.2.Design Science Research Methodology Process

Peffers et al. (2007) design a six-step process Design Science Research Methodology as shown in **figure 3**. The first step in this process is to identify the research question. In the second step a solution is defined based on the specific problem. The third step is focused on designing the artifact. In the fourth step they evaluate the effectiveness of designed artifact to solve the research problem by demonstrating examples or case studies. In the next step, they compare the actual of using the artifact (from the previous step) to the objectives. The last step helps researchers and professionals to get a clear understanding of the artifact and its usability.

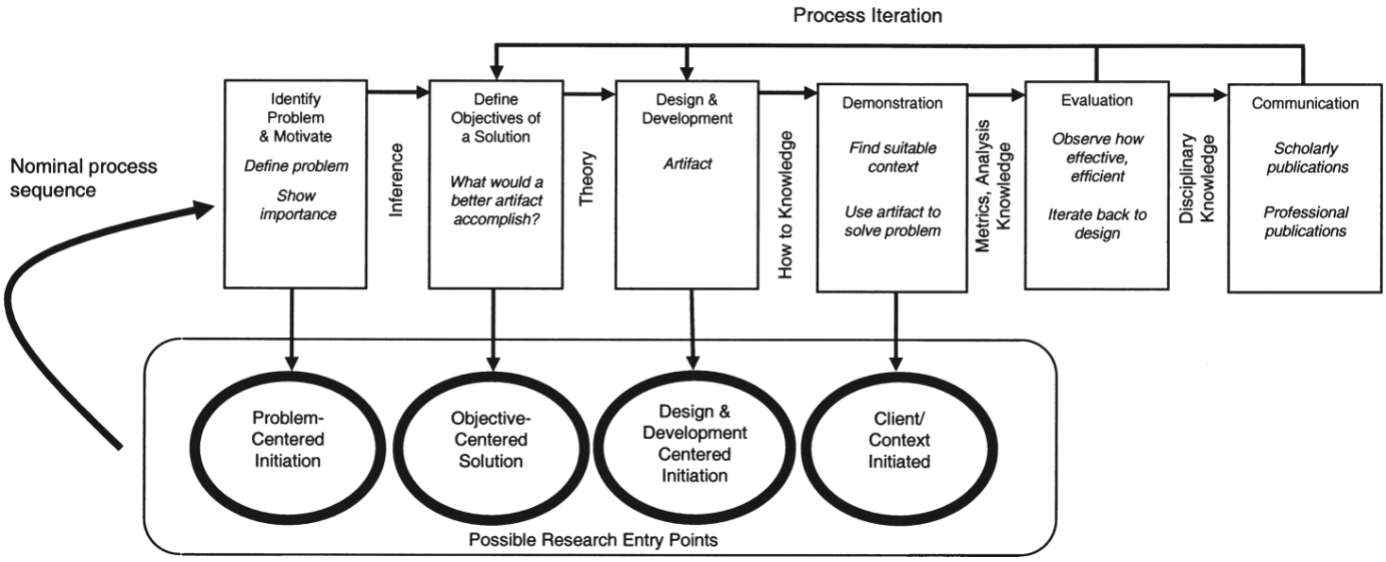


Figure 3. Design Science Research (DSR) Process Model (Peffers et al., 2007)

According to Peffers et al. (2007), researchers don't have to always follow the steps in the suggested order, however, it is important to go through all the steps. The outcome of DSR is always a practical artifact. This artifact can be a product, a model, a technique, a tool or even a process (Venable et al., 2017).

Following the steps mentioned above, we use DSR for our research as described below.

1. **Identify Research Problem and Motivation:** We define our research problem (i.e., research question), which is comparing the level of complexity between microservice orchestration and choreography in the development of e-commerce applications, based on

the most recent studies in the literature review and real-world e-commerce scenarios in the industry. The lack of a definitive answer for businesses using e-commerce to choose the best composition style for their microservice-based applications motivates us to focus on the defined research question.

2. **Define Objectives of a Solution:** Next, we propose solutions to the problem by defining the objectives. Our objectives are focused on evaluating and comparing the level of complexity for two microservice composition techniques (choreography and orchestration) using BPMN 2.0 executable models in the domain of e-commerce. This evaluation can help e-commerce companies get a better understanding of which microservice composition style best suites their business needs.

3. **Design and Development:** In the third step, we develop our e-commerce scenarios. To this end, we follow a series of steps: 1. Study e-commerce websites; 2. Identify, document, and classify e-commerce scenarios into 3 categories (small, mid-sized, end-to-end); 3. Use BPMN 2.0 to model choreography and orchestration for each scenario from the 3 categories.

4. **Demonstration:** We use Zeebe BPMN Modeler, Zeebe-docker, CAMUNDA automation engine (Zeebe.io, n.d.), and Amazon Web Services-AWS cloud to design (i.e., model the workflows or microservice compositions and deploy the microservices on the cloud. We do this for all the scenarios we selected earlier. In our development process, we follow the BPMN 2.0 modelling guidelines in https://stage.docs.zeebe.io to model our workflows into microservice choreographies and orchestrations, representing the communication mechanisms among microservices and characteristics of each composition style.

5. **Evaluation:** In this step, we evaluate the models by making sure that the applications generated by the development tool from the models accomplish what they are supposed to, and then and by assessing the complexity of the models. In the first step, we validate our models using the XML code generated by the modeling tool from the models. We import our models into the cloud for deployment, create and complete workflow

instances. Then, we evaluate the workflow models using three tools called Zeebe Simple Monitor, Kibana Elasticsearch cluster to visualize log files, and Camunda Operate to test the instances in the workflows and execute them to make sure they run with no errors. In the second step of the evaluation, we use complexity metrics from the software engineering literature (for details on the complexity metrics, please refer to Chapter 5) to assess and compare our models and draw insights.

In this research, we follow the evaluation classification proposed by Hevner et al. (2004) to evaluate our models**. Table 4** shows the five different methods of evaluation for design science research.

Table 4. Methods of evaluation (Hevner et al., 2004)

Methods	**Description**
Observational	Includes Case Study and Field Study
Analytical	Includes four components: Static Analysis, Architecture Analysis, Dynamic Analysis, and Optimization
Experimental	Includes Controlled Experiment and Simulation
Testing	Includes Functional (black box) testing and Structural (white box) testing
Descriptive	Includes informed argument and scenarios

We use a combination of ***Descriptive*** and ***Experimental*** methods to evaluate our models. This is done by proposing illustrative scenarios, which focus on real-world e-commerce applications to develop BPMN 2.0 workflows. Furthermore, we adopt technical experiment as a method to evaluate the performance of each designed artefact via adapting complexity as a metric to each workflow to compare the level of complexity in each microservice composition approach.

6. **Communication:** We use the results of our evaluation to analyze both composition techniques and use the results to discuss the impacts of complexity on the modifiability and understandability of the models. Finally, we discuss the conclusions and suggest areas for future research.

4 Chapter 4. Building BPMN Models

In this chapter, we discuss the process of designing, deploying, and executing our BPMN models. First, we explain how we design our models using Zeebe Modeler (Zeebe.io, n.d.) and give a brief description of some underlying concepts of BPMN used in our models. Second, we describe the modeling and simulation (i.e., simulated execution) flow of the designed choreography and orchestration models categorized by the size of the workflow (small-sized, mid-sized and the end-to-end). In section 4.3, we discuss how we use Zeebe Simple Monitor (Zeebe.io, n.d.) to deploy the resulting models for both choreography and orchestration, and create and execute instances of each model following the guide on https://stage.docs.zeebe.io.

4.1. Create a BPMN model using Zeebe Modeler

This section focuses on designing a set of models used as artifacts for this research. We use BPMN techniques to design multiple models. Our main modeling tool is Zeebe Modeler, which is an open-source desktop modeling tool that supports BPMN 2.0 (Zeebe.io, n.d.). Zeebe modeler also generates XML code for each BPMN model, which we use for one of our metrics (Line of code) to measure the complexity of the models (see Chapter 6).

4.2. Selection of E-commerce scenarios

All the workflows used in this research are based on realistic e-commerce scenarios. We divide the abovementioned scenarios into three categories:1-Small size; 2-Mid-size; and 3-End-to-end. We will use these categories later to evaluate the impact of the size of the models on their level of complexity. Each model is intended to illustrate one specific e-commerce process such as payment, shipment and etc. We will further discuss the scenarios in Chapter 6.

Below are the steps we take for the selection of our workflows:

1- Define scenarios inspired from real e-commerce processes
2- Begin by designing small workflows which use one single service
3- Combine small workflows to design mid-sized models

4- Use the insights from the predefined sets of workflows to create a big model which focuses on an end-to-end e-commerce process.

4.3. Selection of BPMN Elements

In general, there are four types of elements used in BMPN modeling as defined below:

1- Flow Objects, which include activities, events, and gateways
2- Connecting Objects, which link two flow objects
3- Swimlane Objects, which include pools and lanes and are used to separate activities into individual sections based on different organizational processes
4- Artifacts, which include data objects, group and annotation and are used as additional elements for a workflow if needed

The models used in this study consist of start and end event, service tasks to represent microservices and sequence flows to connect those service tasks to each other. We use XOR gateways to show alternative flows where only one option is executed. We use message events for all the communications involved in each workflow.

4.4. Designing BPMN-Based Workflows for Choreography and Orchestration

In this section we describe the design and simulation flow of our models using Zeebe Modeler BPMN 2.0 and Zeebe Simple Monitor(Zeebe.io, n.d.). Each model depicts one single service used in the domain of e-commerce. We compose each workflow using both composition styles (orchestration and choreography). These composition techniques use different communication mechanisms, therefore for orchestration of our BPMN model designs, we use the Intermediate message catch event which uses a send/receive message approach managed by the orchestrator to call each service and should wait for a response from the service to be able to call the next service. Whereas for choreography, we implement event choreography which uses service calls as events. This way services work independently without depending on calling other services.

4.4.1. Small-sized BPMN Workflows

4.4.1.1.User Authentication

This service is used to authenticate users on the e-commerce website. If the user is already registered, then the system redirects them to a login page where they can enter their login credentials. If the user is new, they are given the choice to either proceed as a guest or register on the website. **Figure 4** shows the choreography workflow for user authentication designed in Zeebe modeler. We define a variable called "userId" with two values Yes/No, which we use in the execution of the workflow. Upon creating a new instance on Zeebe Simple Monitor, the first task is called (Browse site), next we move to the XOR gateway to choose between the alternatives. For that we should call the userId variable and give it a value based on the scenario we are replicating. In our execution, we choose Yes for the XOR gateway, and this triggers the log in task to complete the process. **Appendix A** shows a snippet of the simulated flow mentioned above with the list of jobs created on Zeebe Simple Monitor. The sequence flows in green show the path for the simulated flow.

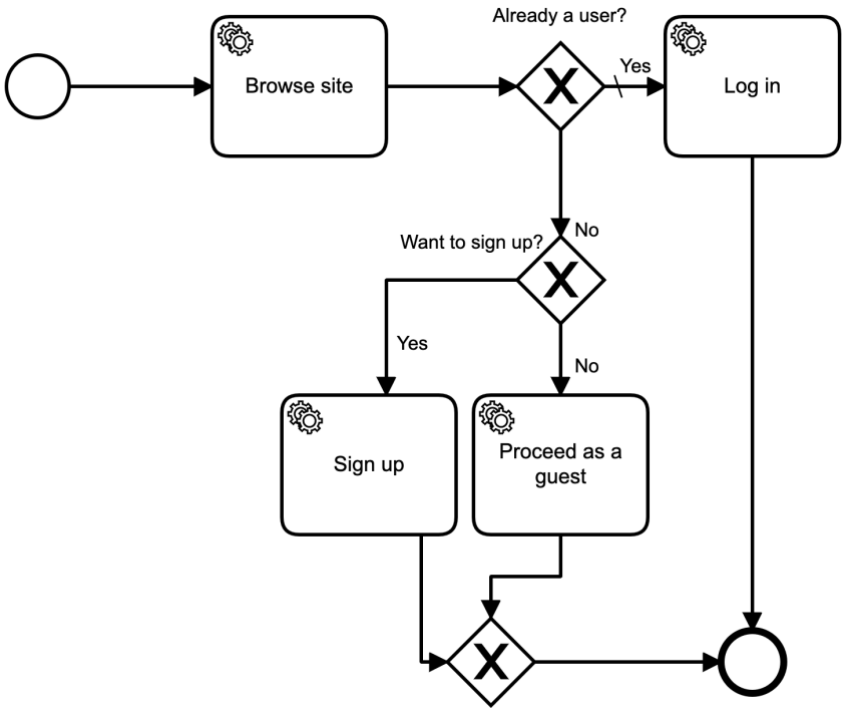


Figure 4. Choreography model for user authentication workflow

The second workflow, as shown in **figure 5**, uses orchestration to model the same scenario. To implement orchestration for this workflow, we have defined a service task called e-commerce, which acts as the orchestrator. In order to deploy and execute the workflow on Zeebe Simple Monitor (Zeebe.io, n.d.), we need to define an automated logic to call the next service task in the process without having to develop any code. Hence, we have defined a variable called serviceCall which the orchestrator will use to call the next service in the process by giving it a predefined value. We use the name of each service task as the values for our defined variable to call that service. All the communications between services are mapped using the Message Intermediate Catch Event, which is a BPMN 2.0 message event. **Appendix B** shows a snippet of this workflow we have executed on Zeebe Simple Monitor (Zeebe.io, n.d.).

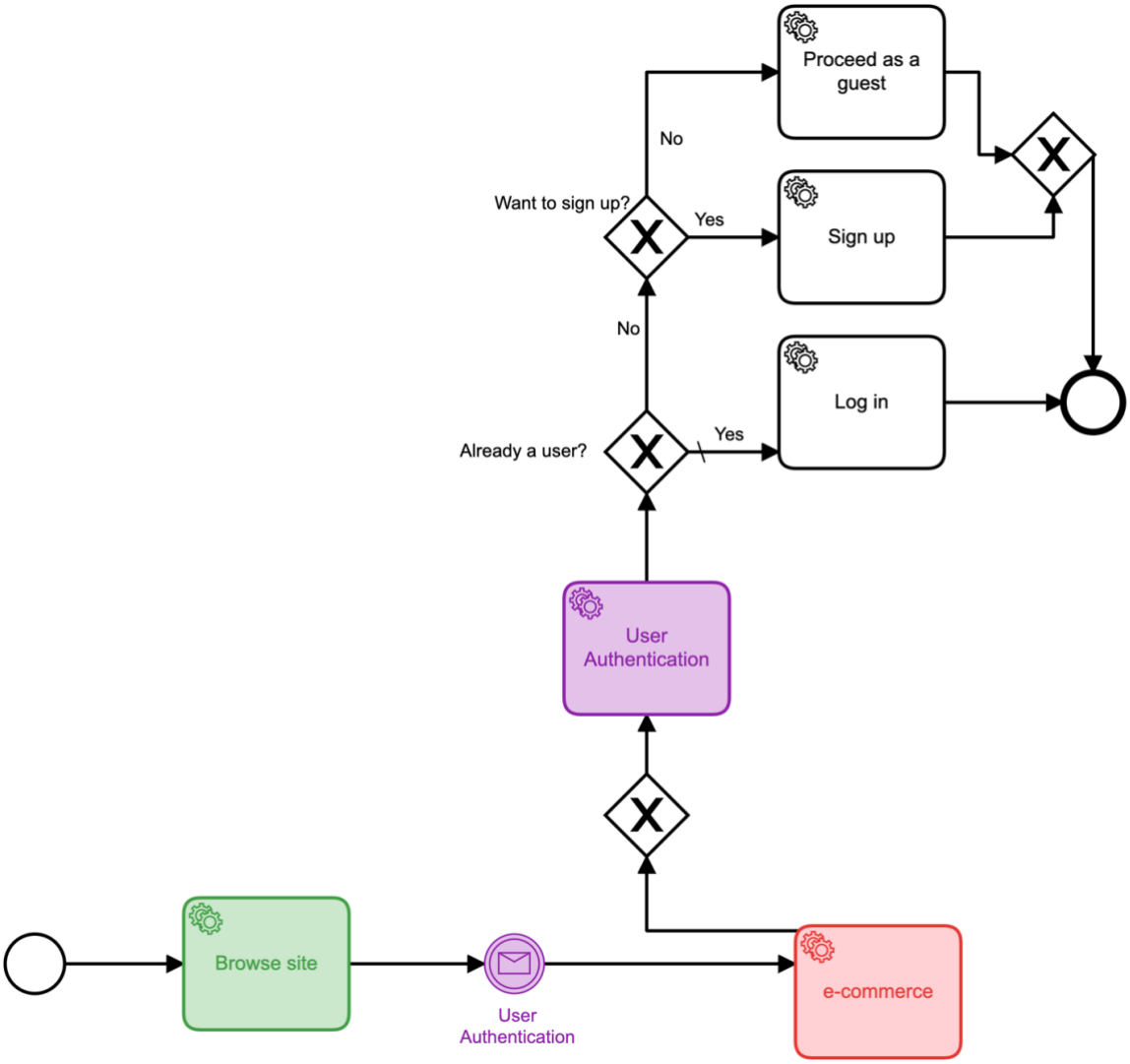
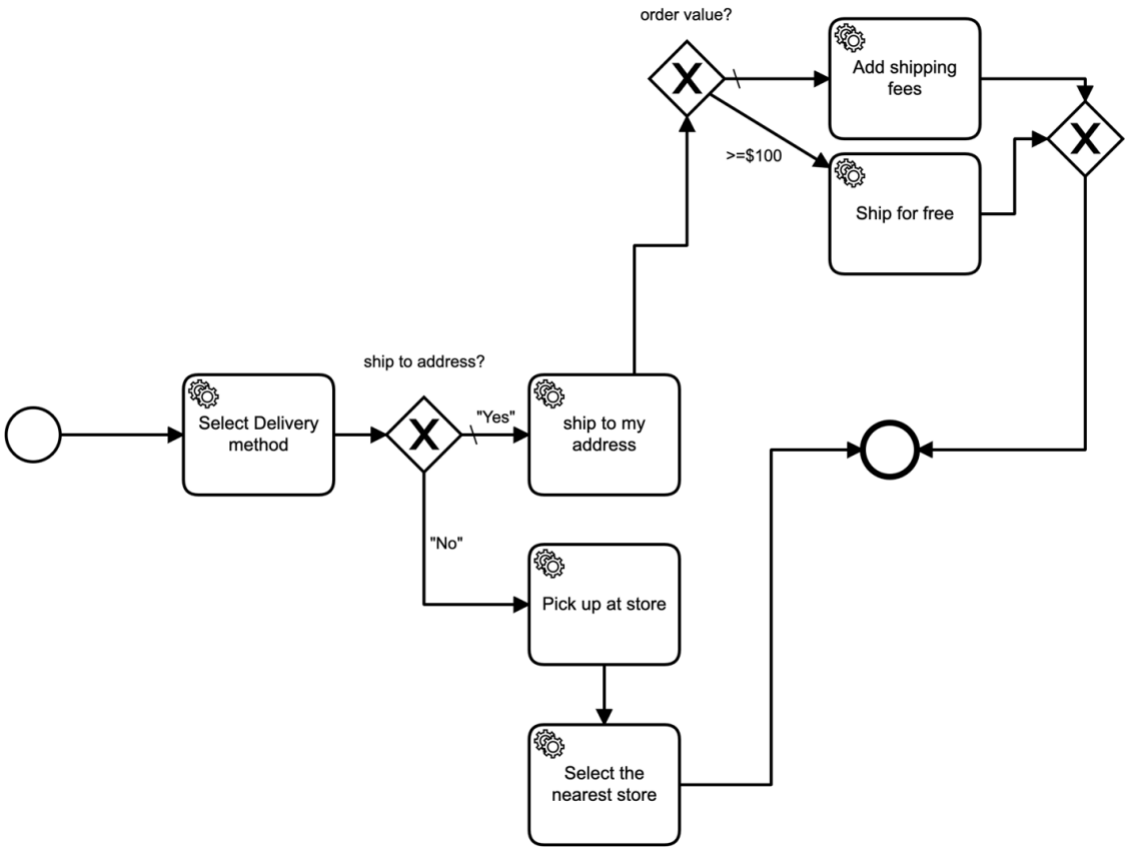


Figure 5. Orchestration model for user authentication workflow

4.4.1.2.Shipment

In this scenario the user should choose between two delivery methods: ship to address or pick up at store. The store pickup (also known as curbside pickup) is a relatively new feature and has been very popular recently due to the current pandemic which has affected the way businesses operate. One more feature that we add to this scenario is the shipping fee option, which is based on the total order value. For orders above $100 users will not be charged any shipping fees. For implementing this we use a feature on Zeebe Modeler to add a condition on the sequence flow. To that end, we define a variable called "orderValue" and add the following condition "=orderValue>=100" to the sequence flow which is linked to the shipping fee service task. For both XOR gateways we have defined variables and values of Yes/No. **Figure 6** shows the shipment workflow as a choreography. **Appendix C** shows the simulated flow we have executed on Zeebe Simple Monitor (Zeebe.io, n.d.) for this workflow.



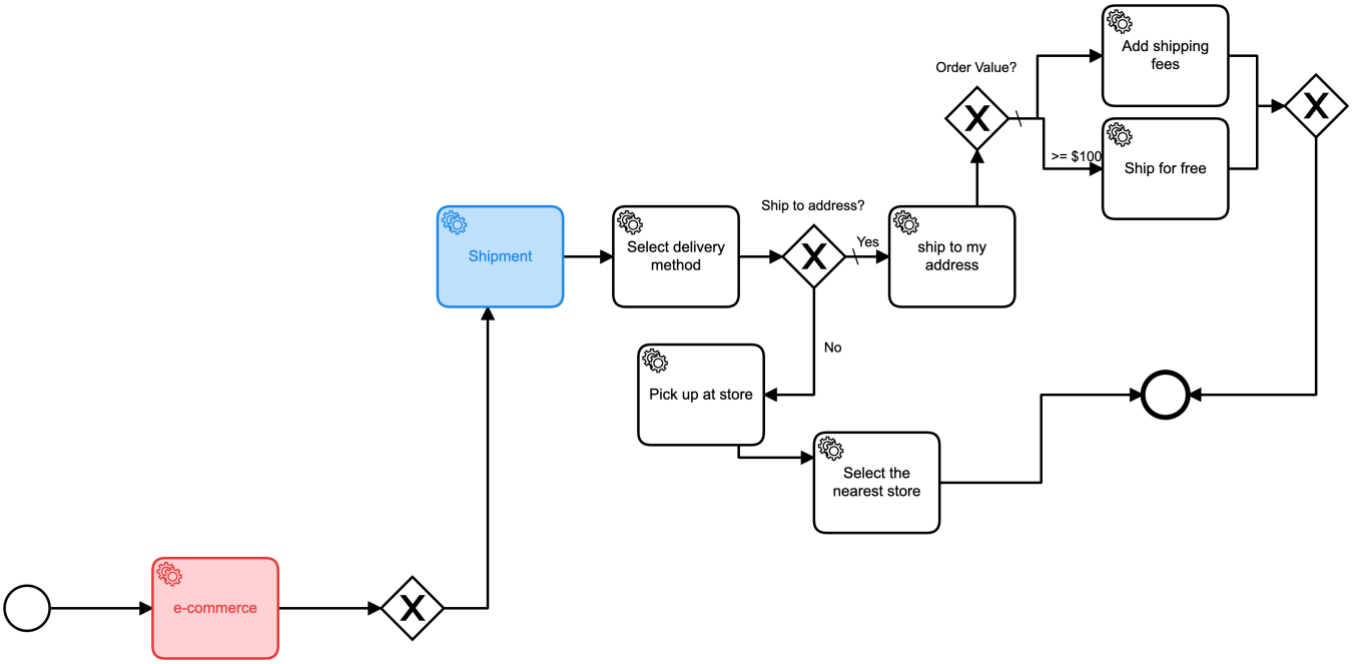


Figure 6. Choreography model for shipment workflow

Figure 7. Orchestration model for shipment workflow

The shipment workflow modeled as an orchestration is shown in **figure 7**. As mentioned above, e-commerce service task is the central controller in this process. **Appendix D** shows the simulated flow we have executed on Zeebe Simple Monitor (Zeebe.io, n.d.) for this workflow.

4.4.1.3. Payment

This service allows users to make payments online using debit or credit. Some websites support another payment method called cash-on-delivery, which enables users to pay for their orders after they receive the items. **Figure 8** shows the designed choreography model for the shipment workflow using Zeebe Modeler. We have defined a variable called "paymentId" with two values Yes/No, which we use in the execution of the workflow. Upon creating a new instance on Zeebe Simple Monitor, the payment service task is called to initiate the payment. We use an XOR gateway to check if the payment has been successfully

processed using the predefined variable. **Appendix E** shows a snippet of the simulated flow mentioned above with the list of jobs created on Zeebe Simple Monitor (Zeebe.io, n.d.). The sequence flows in green show the path for the simulated flow.

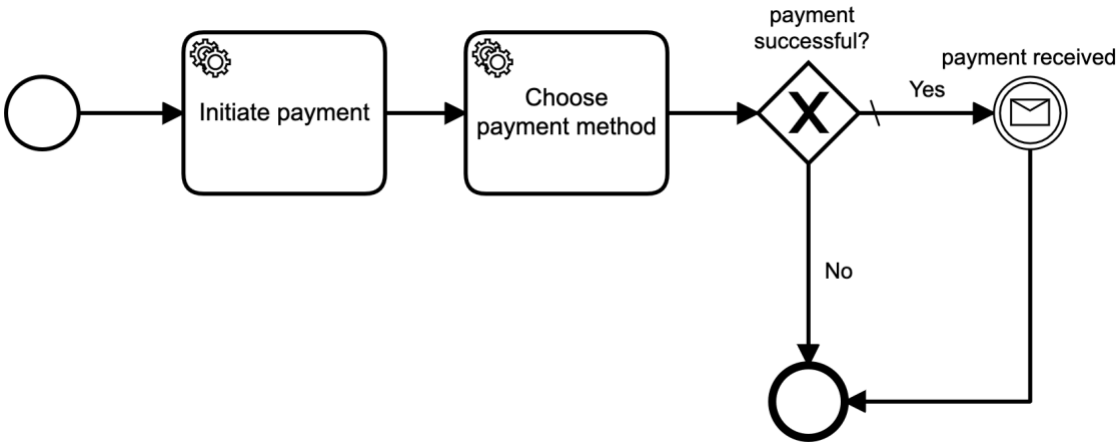


Figure 8. Choreography model for payment workflow

The payment workflow modeled as an orchestration is shown in **figure 9**. As mentioned above, e-commerce service task is the central controller in this process. **Appendix F** shows the simulated flow we have executed on Zeebe Simple Monitor (Zeebe.io, n.d.) for this workflow.

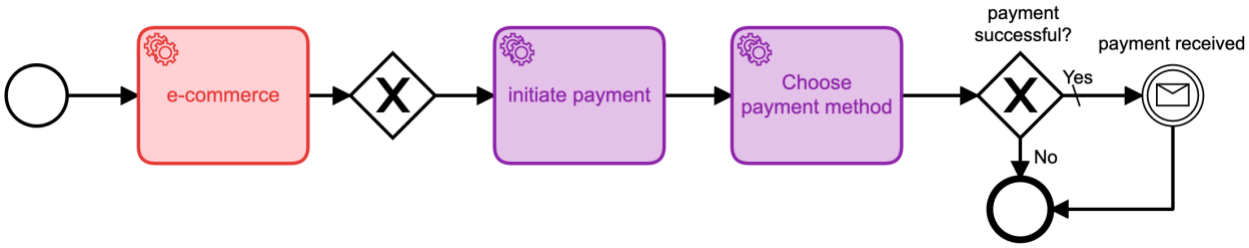


Figure 9. Orchestration model for payment workflow

4.4.2. Mid-sized Workflow

In this section, we use the workflows described above to design a new set of models, which include more than one service in their process. We use the same tools for the design, deployment, and execution of the models (Zeebe.io, n.d.).

4.4.2.1.User Authentication + Shipment

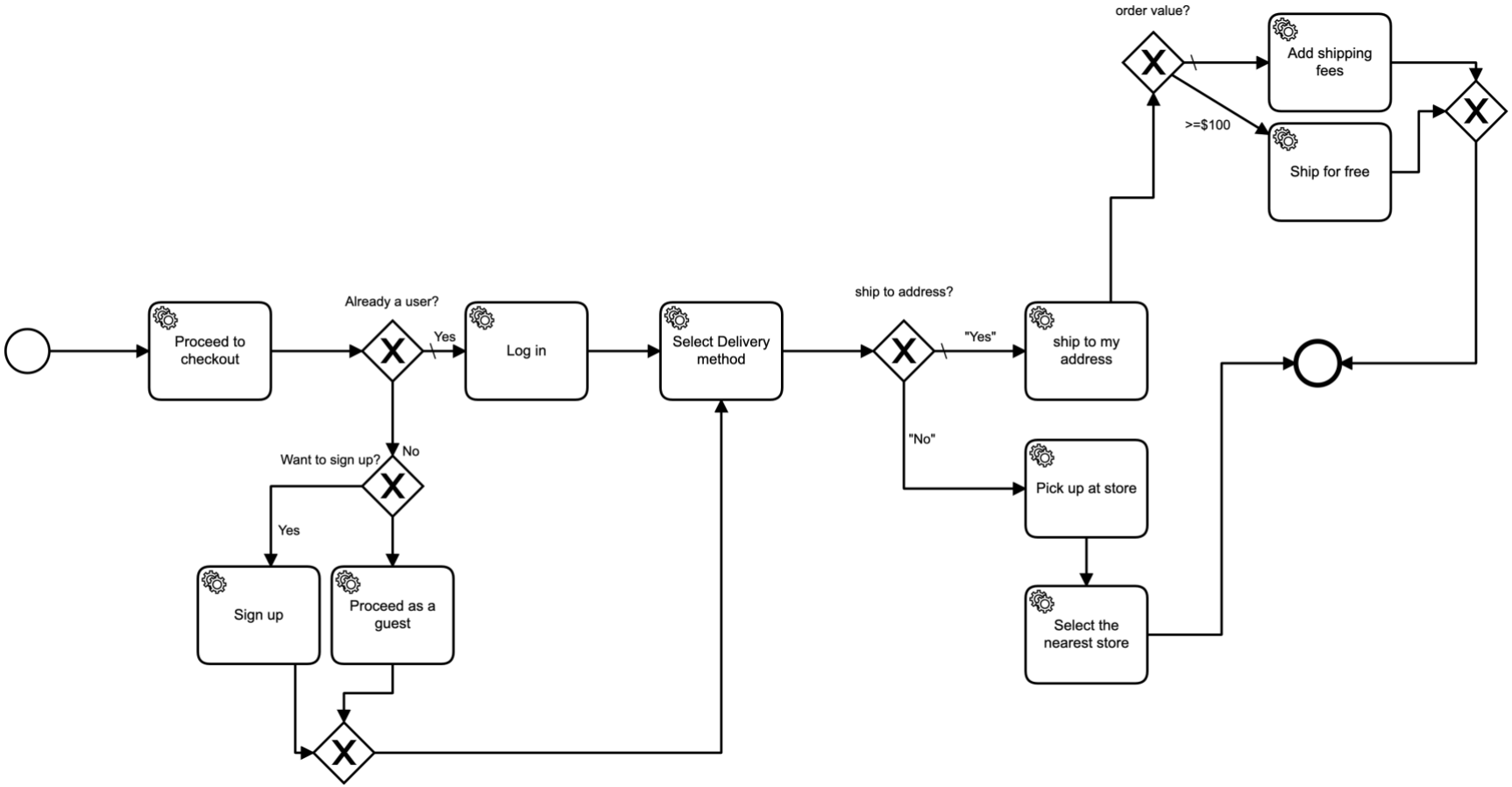


Figure 10. Choreography model for user authentication +shipment workflow

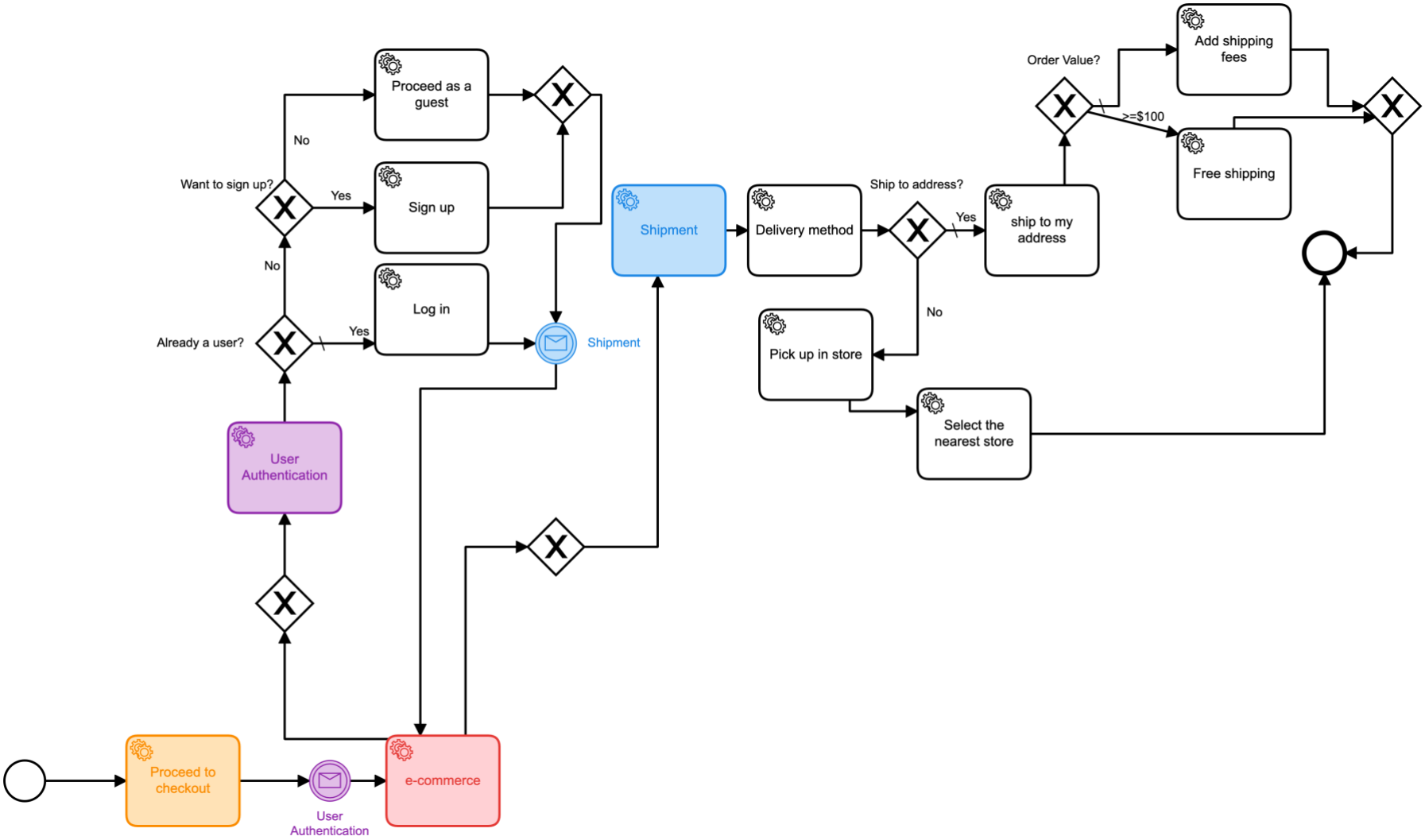


Figure 11. Orchestration model for user authentication + shipment workflow

4.4.2.2.Shipment + Payment

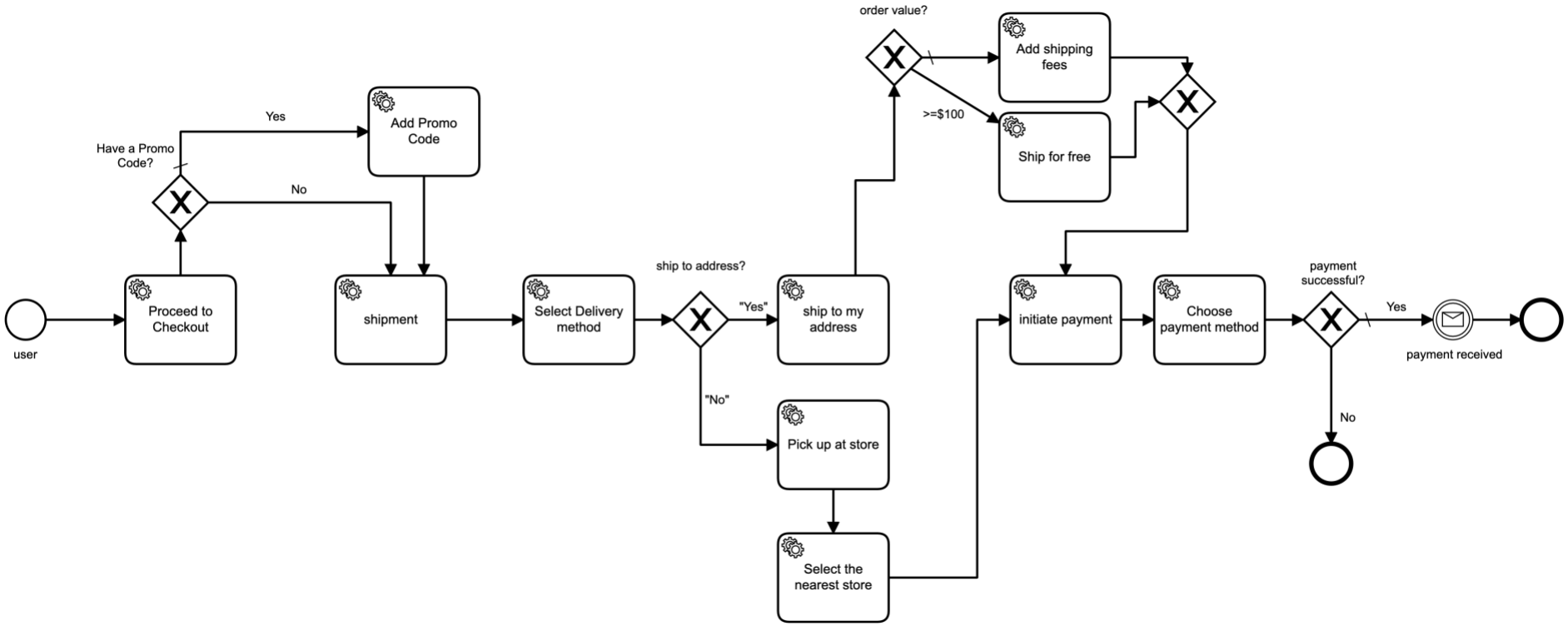


Figure 12. Choreography model for shipment + payment workflow

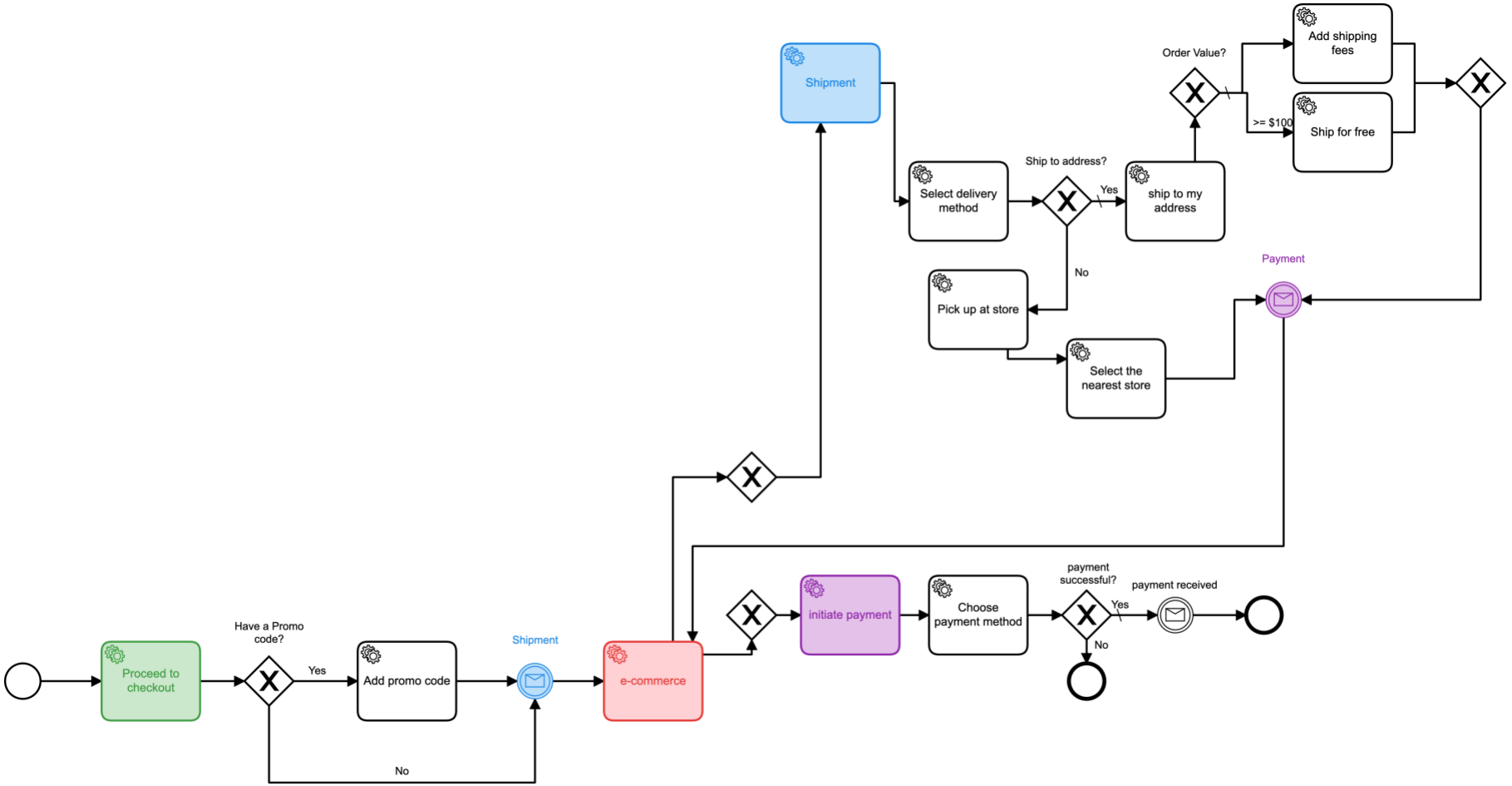


Figure 13. Orchestration model for shipment + payment workflow

4.4.3. End-To-End Workflow

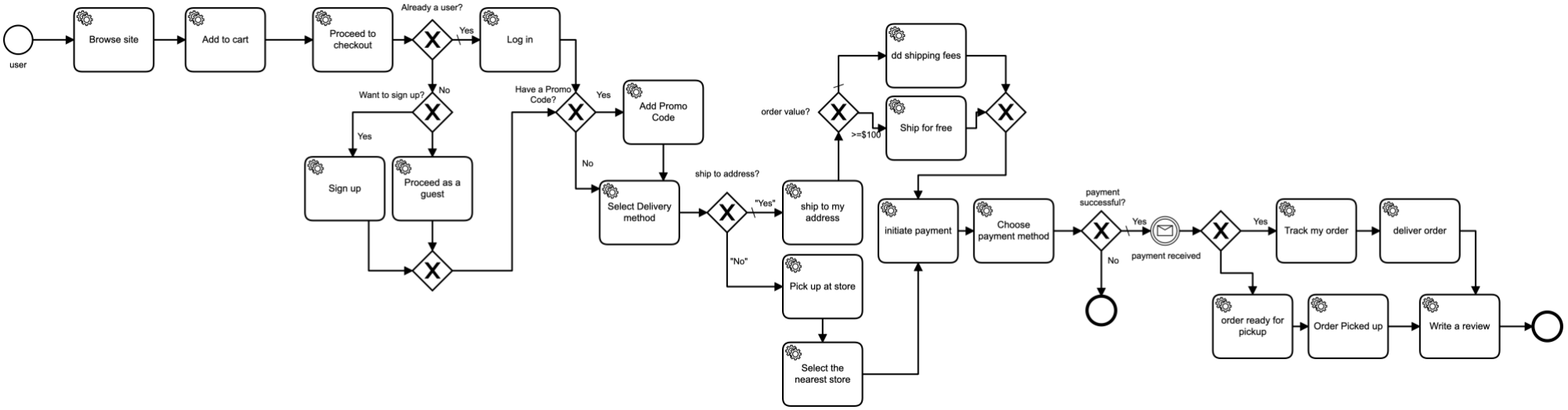


Figure 14. Choreography model for end-to-end workflow

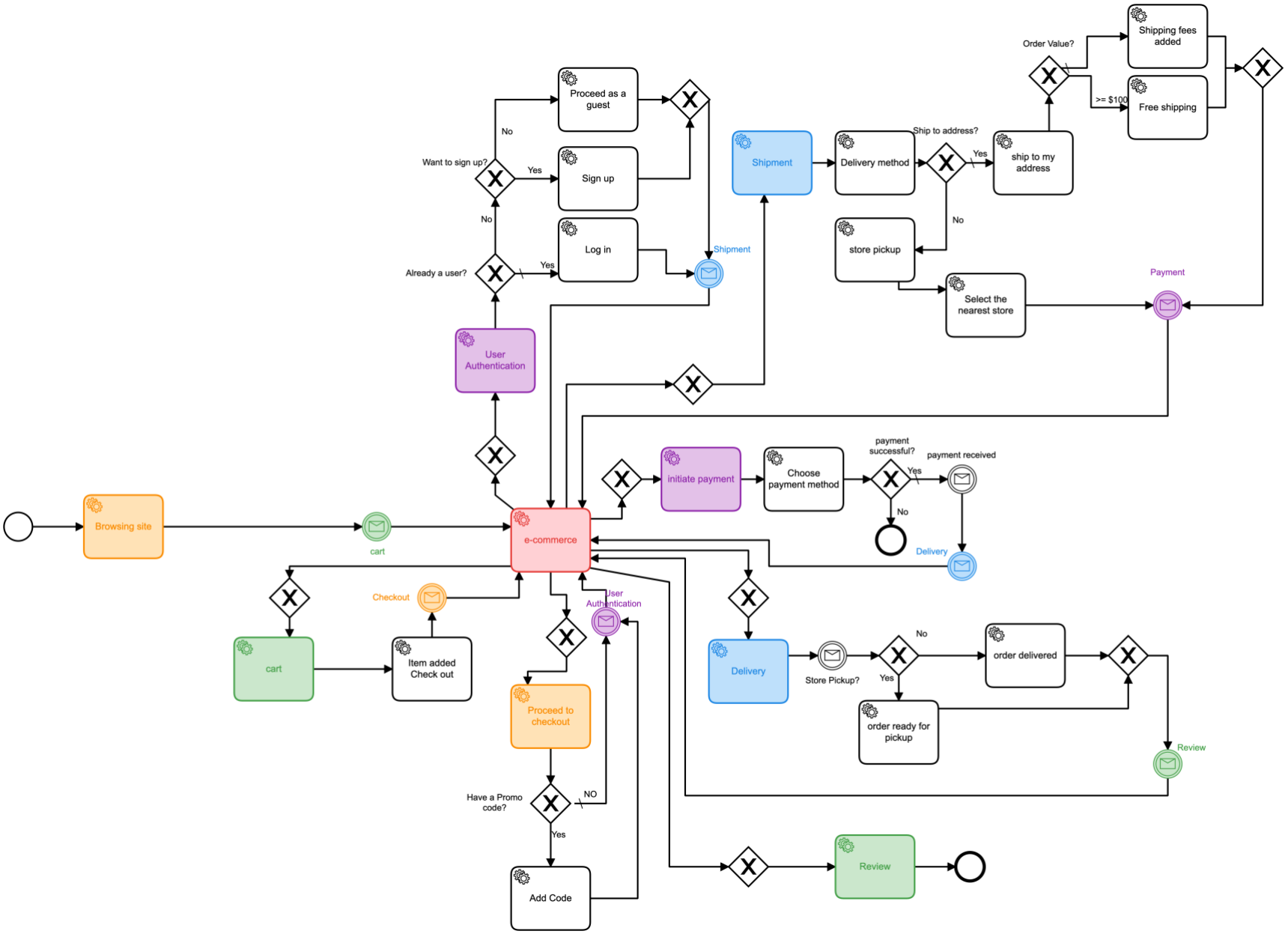


Figure 15. Orchestration model for end-to-end workflow

4.5.Deploy BPMN Workflows

After we design our models on Zeebe Modeler, we use Camunda Workflow to deploy our designed models (Zeebe.io, n.d.). Zeebe Simple Monitor is the platform we use to add new deployments and run instances of each workflow. It is a platform that gives access to a consistent environment to deploy and run BPMN models, using AWS cloud as the server to install Docker. On the client side we install Zeebe installer on either a Mac or Windows system. To use Docker, we run the following command in the AWS cloud:

docker run --name zeebe -p 26500-26502:26500-26502 camunda/zeebe:latest

After Zeebe Docker is configured, we use server IP address 54.203.144.XX to access the Docker. We are deploying our workflows using port 8082. To make our deployments accessible to all the team members involved in this research, we use server IP address instead of the localhost. We also need to make our IP address private to block an unauthorized access to our deployed models.

In order to upload each model, we need to add new deployments on Zeebe Simple Monitor (Zeebe.io, n.d.). The selected workflows can be uploaded if there are no errors in the design. If there is anything wrong in the design of the workflow, the deployment fails. All the workflows proposed in this research have been tested and have been successfully deployed.

After the workflows are deployed on Zeebe Simple Monitor (Zeebe.io, n.d.), they are given an auto-generated workflow key. To execute a workflow, we need to create a new instance of that workflow. Each instance is considered as a single execution of the workflow. To complete an active instance, we need to complete a job. Jobs are created for each service task involved in a workflow, and in order to perform a task we need to complete a job which is managed by an in-built function called job worker (Zeebe.io, n.d.). According to the Zeebe documentation on https://stage.docs.zeebe.io, a job worker is a long process which is constantly trying to activate jobs for a given task and completes them after executing its business logic. Depending on the workflow, some jobs require us to call a variable that we have defined in our design and give it a predefined value to be able to proceed with the next task. If we enter a wrong variable or value, an error will be posted under the incidents tab where you can check all the errors that occurred when completing

each job with a description of the error. Once the issue is resolved we can proceed to the next task in the workflow.

5 Chapter 5. Complexity Measurement of BPMN Models

This chapter discusses the existing complexity metrics employed in software engineering and how they are adapted to the domain of Business Process Modeling, which is the essential cornerstone of our research.

5.1. State of the art in complexity metrics

There have been a lot of studies on measuring and evaluating the quality of software products using different metrics. Conte et al. (1986), defined five design principles that can be used to measure the quality of software design. These principles include:

1- **Coupling:** describes the interconnections among the modules.
2- **Cohesion:** describes the relationships between the elements of a module.
3- **Complexity:** describes the number and size of the control constructs.
4- **Modularity:** describes how modular the system is. In other words, whether the components of the system can be separated and put back together via logical partitioning.
5- **Size:** describes the entire dimension of the software product.

Among these five principles, complexity has been the focus of many studies in the domain of software engineering. According to Kluza et al. (2014, p. 5), "IEEE Standard Computer Dictionary defines complexity as the degree to which a system or component has a design or implementation that is difficult to understand and verify".

5.2. Similarities between metrics in software engineering and Business Process Modeling

Software engineering metrics have been used to evaluate different features of the software such as error prediction (Wang et al., 2011), measuring the quality of software processes (Rolón et al., 2006), measuring software functional size (Monsalve et al., 2011), and quality metrics in software design (Vanderfeesten et al., 2007). Companies use these metrics to measure the performance of their software products.

Business processes are another important element in the lifecycle of a software product as they are used from the early stages of a project by both software engineers and business analysts to document and gather system requirements (Monsalve et al., 2011) Hence, several studies have focused on finding similarities between software programming and business process modeling. Vanderfeesten et al.(2007) compared the similarities between software programs and business processes based on the modules, elements and compositional structure used in both domains, as shown in **Table 5** (Vanderfeesten et al., 2007).

Table 5. Similarities between software programs and business processes (Vanderfeesten et al., 2007)

Software Programming	Business Process modeling
Module/Class	Activity
Method/Function	Operation
Variable/Constant	Data element

5.3. Complexity metrics measurements approaches

Business processes cannot be measured by only one single metric. Therefore, many studies suggest different measurement metrics for business processes, which are inspired by the ones used in software engineering. In this section, we discuss the state-of-the-art on complexity metrics in business process modeling.

5.3.1. Lines of code metrics

It is a simple method which counts the number of lines of code in software programs. The logic behind LOC is that the length of the code can have a direct link with numbers of errors in the code, reliability and maintainability of the application (J Cardoso et al., 2006). Cardoso et al. (2006) use LOC to adapt three size metrics for BPM:

1- **NOA** = Number of Activities in a workflow
2- **NOAC** = Number of Activities and Control-flow in a workflow
3- **NOAJS** =Number of Activities, Joins, and Splits

The advantage of these size metrics compared to the LOC metric is that they are not language dependent and are easier for users to understand as they are used to measure business processes. However, they may vary based on how process design is conducted and what modeling approach is being used. In other words, a poor process design can lead to a large number of activities which will affect the results of NOA measurement (J Cardoso et al., 2006).

5.3.2. The Control-Flow Complexity metrics (CFC)

Jorge Cardoso (2005) proposed a Control-Flow Complexity (CFC) to measure the level of complexity of business processes. They use this metric to help designers to create models with less complexity which affects the understandability of the business process models. This metric is measured based on XOR-splits, OR-splits, and AND-splits in one process (Jorge Cardoso, 2005). CFC is associated with the number of mental states in a business process as J Cardoso et al. (2006, p.120) explain: "Splits introduce the notion of mental states in processes. When a split (XOR, OR, or AND) is introduced in a process, the business process designer has to mentally create a map or structure that accounts for the number of states that can be reached from the split". Below we explain the mathematical computation of CFC and how these splits (XOR, OR, or AND) are calculated in CFC. **P** stands for **Process** and **A** means **Activity** (J Cardoso et al., 2006).

CFC(P)=

$$\sum_{A \in P, A \text{ is a } XOR-split} CFC_{XOR}(A) + \sum_{A \in P, A \text{ is a } OR-split} CFC_{OR}(A) + \sum_{A \in P, A \text{ is a } AND-split} CFC_{AND}(A)$$

- $CFC_{XOR-split}$(A)= fan − out(A). The control-flow complexity of XOR- splits is determined by the number of outgoing branches that emerge after each XOR split.
- $CFC_{OR-split}$(A)= fan−out(A)−1. The control-flow complexity of OR-splits is determined by the number of outgoing branches that emerge after each OR split.

- $CFC_{OR-split}$(A)= fan−out(A)−1. The control-flow complexity of OR-splits is determined by the number of outgoing branches that emerge after each AND split.

5.3.3. The McCabe's cyclomatic complexity metric (MCC)

Mccabe (1976) proposed the cyclomatic approach to measure the complexity of a software program based on the number of paths through the program. It is one of the most popular metrics in the domain of software given that it can be used to measure complexity for any programming language and format (Banerjee, 2018). MCC is a graph-theoretic technique that calculates the cyclomatic number of a graph by counting the maximum number of linearly independent paths in the graph. This approach was initially proposed to allow developers to control and manage the size of the modules in their programs by measuring the cyclomatic complexity for each module (Mccabe, 1976). He defined the cyclomatic complexity as:

MCC= e - n +2

Where **e** is the number of edges, and **n** is the number of nodes in the graph. The nodes represent blocks of code in a program and the edges are the arcs which connect two nodes (Banerjee, 2018; Mccabe, 1976).

5.3.4. Durfee square metric (DSM) and perfect square metric (PSM)

These two metrics are intended to help designers to measure and control the complexity of their models by calculating DSM and PSM during the design. They are popular for being simple and easy to understand. Despite their simplicity, these metrics consider all the process elements and their frequency in the model, while other simple metrics only focus on one single element. For example, NOA only measures the total number of activities used in a model and Diameter only takes into account the connection (flow) in a model. One other advantage of these two metrics is that they are intended to measure any process models, particularly BPMN models (Kluza et al., 2014; Kluza & Nalepa, 2012).

Kluza et al. (2014, p.11) define DSM and PSM as follows:

"**DSM** equals d if there are d types of elements which occur at least d times in the model (each), and the other types occur no more than d times (each)."

"**PSM** is the (unique) largest number such that the top p types occur (together) at least p^2 times, given a set of element types ranked in decreasing order of the number of their instances."

5.3.5. Information flow metrics by Henry and Kafura

Henry & Kafura (1974) suggest measuring data flow in and out of the system components to define the level of complexity and coupling for procedures and modules used in large-scale software systems. Their approach focuses on evaluating the procedure complexity (PC) based on the frequency of calls in and out of the modules in one system and is defined as (Banerjee, 2018; Henry & Kafura, 1974):

$$\boldsymbol{PC = Length * (Fan - in * Fan - out)^2}$$

The length is calculated using the LOC metric, and the fan-in and fan-out indicate the number of calls to and from each module based on the data flow in the system. The in calls happen when a module is called to perform a task and the out refers to when the module completes the given task (Banerjee, 2018).

J Cardoso et al. (2006) adapt this approach to measure complexity in business processes. Their approach is called the interface complexity (IC) which uses activities in a BP as the modules in object-oriented programming, and the fan-in and Fan-out are the incoming and outgoing control flow connectors of each activity. For the length of the process, they define activities as either black boxes or white boxes. If the activities are defined as black boxes, their length cannot be measured, and the length is assumed to be 1. However, the length of a process where the activities are defined as white boxes can be measured based on the length of the source code for the activities using metrics such as LOC (Banerjee, 2018; J Cardoso et al., 2006). They define the IC metric as:

$$IC = Length * (number\ of\ inputs * number\ of\ outputs)^2$$

5.3.6. The coefficient of network complexity metric (CNC)

This metric is used to measure the complexity of a model based on the number of nodes and arcs involved in the process. As Richard A. Kaimann (1974) defined, CNC is a simple and understandable metric to measure the level of complexity, which is directly affected by the number of arcs. So, the process can become more complex as the number of arcs increases in the process. CNC is calculated as the total number of arcs divided by nodes (activities, joins, and splits); and is generally defined as follow (Banerjee, 2018; Kaimann, 1974; Latva-Koivisto, 2001) :

$$CNC = \frac{Number\ of\ arcs}{Number\ of\ activities, joins, and\ splits}$$

5.3.7. Connectivity level between activities (CLA)

The intention of this metric is to evaluate the impact of structural complexity of BPMN models on their maintainability using a set of defined metrics. These metrics are divided into two categories of Base and Derived measures. The base measures are determined by counting all the different elements in a BPMN model, such as the total number of tasks (NT) in a model (Rolón et al., 2006). Derived measures are inspired from simple base metrics to indicate the number of occurrence between the elements used in a model (Banerjee, 2018).

Rolón et al. (2006) propose multiple base and derived metrics for measuring the level of complexity in BPMN models. One of the derived metrics they propose is called the CLA metric which is measured by counting the total number of activities (TNA) divided by the total number of sequence flows between activities (NSFA) (Rolón et al., 2006).

$$CLA = \frac{TNA}{NSFA}$$

TNA is a derived metric and is composed of two base metrics added together. These metrics are the total number of tasks (TNT) and the total number of collapsed sub-processes (TNCS) in a model (Rolón et al., 2006).

$$\boldsymbol{TNA = TNT + TNCS}$$

5.3.8. Halstead-based process complexity

The Halstead metric is a popular measurement technique in software domain which is intended to measure the complexity of software by counting the number of all the operators (numerical operators) and operands (variables and constants) in the software program. The results of the measurement are used to evaluate three software properties: software length; software volume; and software difficulty (Solichah et al., 2013). This metric is composed of four measures (n1, n2, N1, and N2), defined below (HALSTEAD, 1977):

- n1= The number of operators such as while, if, etc.
- n2= The number of unique operands including variables and constants
- N1= The total number for the frequency of operators
- N2= The total number for the frequency of operands

J Cardoso et al. (2006) adapt this metric to measure the complexity of business process models and use the same measures (n1, n2, N1, and N2) to evaluate process length; process volume; and process difficulty. It is calculated as follow (Banerjee, 2018):

- n1= The number of activities, joins, splits, and other control flow elements in a BP
- n2= The number of data containers used by the process and activities
- N1=The total number for the frequency of type n1
- N2= The total number for data containers (n2)

5.3.9. Structural metrics

Structural metrics are used to measure the level of understandability and error-probability in a model. Structural metrics evaluate the size or complexity of business processes and are

presented as internal quality indicators (Sánchez-González et al., 2012). According to Sánchez-González et al. (2010) structural metrics help get a better understanding of business processes which improves the quality of the models. They propose a number of square metrics, shown in **Table 6**, to evaluate the complexity of BPMN models considering their correlations with the understandability and modifiability of the models. They depict the correlations as either positive (+) or negative (-) for each metric (Sánchez-González et al., 2010).

Table 6. Structural metrics for process models (Sánchez-González et al., 2010)

Metric	**Correlation**	**Description**
Number of nodes	Negative	Total number of activities and routing elements in a model
Diameter	Negative	The length of the longest path from a start node to an end node
Density	Negative	The ratio of the total number of arcs to the maximum number of arcs
The Coefficient of Connectivity	Negative	The ratio of the total number of arcs in a process model to its total number of nodes
The Average Gateway Degree	Negative	The average number of both incoming and outgoing arcs of gateway nodes in the process model
The Maximum Gateway Degree	Negative	The maximum sum of incoming and outgoing arcs of these gateway nodes
Separability	Positive	The ratio of the number of cut-vertices on the one hand to the total number of nodes in the process model on the other
Sequentiality	Positive	The degree to which the model is constructed out of pure sequences of tasks.
Depth	Negative	The maximum nesting of structured blocks in a process model
Gateway Mismatch	Negative	The sum of gateway pairs that do not match with each other, e.g., when an AND-split is followed by an OR-join
Gateway Heterogeneity	Negative	Different types of gateways used in a model
Cyclicity	Negative	The relation between the number of nodes in a cycle to

		the sum of all nodes
Concurrency	Negative	The maximum number of paths in a process model that are activated due to AND-splits and OR-splits

One of the hypotheses proposed by (Sánchez-González et al., 2010) explains the effects of diameter and the coefficient of connectivity (CoC) on the level of understandability of BPMN models which we will discuss in the final chapter of this study. Their hypothesis suggests the level of difficulty of BPMN models based on (Sánchez-González et al., 2010):

- The number of nodes
- The path from start to end
- The number of nodes connected to decision nodes
- The gateway heterogeneity

6 Chapter 6. Complexity Measurement of Choreography and Orchestration

In the previous chapter, we discussed the state of the art on the complexity metrics in the domain of software engineering and business process modeling. In this chapter we adapt some of those metrics, which are applicable to our BPMN models, designed in Chapter 4, and use them to measure the complexity of each model. This chapter consists of two sections. In section **6.1** we use Shipment workflow (from Chapter 4) as a sample to show how we apply the complexity metrics discussed in the previous chapter. Next, we apply the metrics on the remaining models to measure their complexity. In section **6.2** we present a table containing the results of applying the metrics on all the designed workflows from the three categories.

6.1.Application of complexity metrics

In this section, we describe how we apply the complexity metrics, discussed in chapter 5 to measure the complexity of our designed models. However, not all the metrics are applicable to our models, as some of them require elements of BPMN that we do not use in designing our models. For instance, the Halstead-based process complexity uses data containers as one of its four measures, but there are no data containers in our models. Hence, this metric cannot be used for our evaluation. We use the shipment workflow, shown in figures 6 and 7 as our sample model to apply the selected complexity metrics.

6.1.1. Lines of code metric

The lines of code metric discussed in section 5.3.1 can be used to measure the complexity of a process. Zeebe modeler generates XML code for each BPMN model. The total number of lines of code for the choreography workflow is 194, while the same workflow modeled as an orchestration generates 244 lines of code. **Appendix G** shows the XML code for both models.

6.1.2. Size Metrics

We apply the three size metrics proposed by J Cardoso et al. (2006) from section 5.3.1 to our sample models and the results are shown in **Table 7**.

Table 7. The size metrics for choreography and orchestration

Choreography	NOA= 6
	NOAC= 18
	NOAJS= 9
Orchestration	NOA= 8
	NOAC= 21
	NOAJS= 12

6.1.3. Control-Flow Complexity Metrics (CFC)

As discussed in section 5.3.2, we apply the CFC metric to the choreography and orchestration sample models. There is only one kind of gateway (XOR) used in the models, therefore the CFC for each workflow equals the total number of CFC_{XOR}. As evident from **figure 6**, in the choreography workflow, there are 5 outgoing arcs in total, so the CFC equals 5. And according to **figure 7**, for orchestration the CFC equals 6, which shows a higher level of complexity in orchestration.

6.1.4. Durfee Square Metric (DSM) and Perfect Square Metric (PSM)

As explained in section 5.3.4, DSM refers to the least number of elements that is used in a workflow. Hence, to measure the DSM metric we list all the elements used in each workflow with the number of times they occurred to find the element with the least frequency. As shown in **Table 8**, the DSM for choreography is lower than orchestration which depicts that there is more complexity in orchestration as the result of this measurement.

Table 8. Element types and their frequency in choreography and orchestration workflow

Element types	**Frequency**	
	Choreography	**Orchestration**
Service Tasks	6	8
XOR Gateway	3	4
DSM	DSM=3	DSM=4

For PSM, we perform the computations by giving an assumed value to p, based on the occurrence of each element in the workflow. We start with p=1, which counts the

frequency of the first element in the workflow, we add up to the value of p and count the combined occurrence of the elements for as long as the total satisfies the boundary condition of p^2 times. For choreography, if we assume p to be 4, the combined occurrence for the elements is 11 which fails to satisfy the boundary condition of at least 16, therefore the PSM equals 3. We perform the same measurement for orchestration, and we get the same result for PSM, which equals 3.

6.1.5. Coefficient of Network Complexity Metrics

As discussed in section 5.3.6, CNC measures the complexity of the workflow by counting the total number of arcs relative to the count of other elements in the workflow. We apply this metric to the workflows in figures 6 and 7 and the results are shown below in **Table 9**. For the choreography model, the total number of arcs (sequence flows) equals 12 divided by the total counts of all other entities, which includes service tasks, OR gateways, start and end events. For the orchestration the total number of arcs is 16 divided by 14. As the numbers show in **Table 9**, orchestration has a higher level of complexity compared to choreography.

Table 9. The Coefficient of Network Complexity metric

CNC	Choreography	$\frac{12}{11} = 1.09$
	Orchestration	$\frac{16}{14} = 1.14$

6.1.6. Structural Metrics

As described in section 5.3.9, structural metrics are inspired by the Coefficient of Connectivity (CoC) metric and focus on measuring the Diameter in a workflow. Hence, in this section we measure the Diameter value for both orchestration and choreography as described in section 5.3.9. As shown in figure 6, the Diameter for choreography is 7 while the Diameter measured for orchestration (figure 7) equals 10. Therefore, our orchestration workflow has a higher Diameter, which means the workflow has a lower

level of understandability and is more error-prone compared to the choreography model (Sánchez-González et al., 2010).

6.2. Comparison of the Complexity Metric Results

We applied all the above-mentioned metrics on all our designed workflows and we summarised the results in **Table 10** below for both choreography and orchestration. As evident from the results, all the metrics, except for CNC, show a higher level of complexity in orchestration processes compared to choreography. The results from the CNC metric show more complexity in choreography of small-sized models compared to orchestration, whereas in bigger models with two or more services involved (mid-sized and end-to-end) the measurements depict a higher level of complexity in orchestration.

Table 10. Comparison of the complexity metric results for choreography and orchestration

BPMN Complexity Metrics	Scenario	Choreography	Orchestration
CNC= Number of arcs/ Number of activities, joins, splits	User Authentication	1.11	1.07
	Shipment	1.09	1.07
	payment	1	1
	User Authentication +Shipment	1.16	1.20
	Shipment+ Payment	1.10	1.12
	End-to-end	1.16	1.21
CFC= CFCXOR-split (A) = fan–out(A)	User Authentication	5	6
	Shipment	5	6
	payment	2	3
	User Authentication +Shipment	8	12
	Shipment+ Payment	9	11
	End-to-end	16	24
Lines of code	User Authentication	164	230
	Shipment	194	243
	payment	108	137
	User Authentication +Shipment	330	455
	Shipment+ Payment	364	493
	End-to-end	613	999
	User Authentication	4	6

Number of activities	Shipment	6	8
	payment	2	3
	User Authentication +Shipment	10	13
	Shipment+ Payment	11	12
	End-to-end	20	23
Diameter	User Authentication	6	10
	Shipment	7	10
	payment	4	6
	User Authentication +Shipment	11	20
	Shipment+ Payment	14	20
	End-to-end	22	49
DSM/PSM	User Authentication	DSM=1 PSM=2	DSM=1 PSM=3
	Shipment	DSM=3 PSM=3	DSM=4 PSM=3
	payment	DSM=1 PSM=1	DSM=1 PSM=1
	User Authentication +Shipment	DSM=1 PSM=4	DSM=2 PSM=4
	Shipment+ Payment	DSM=1 PSM=4	DSM=3 PSM=4
	End-to-end	DSM=1 PSM=5	DSM=8 PSM=5

7 Chapter 7. Conclusion and Future Work

7.1.Discussion

This research aims to compare the complexity of the two microservice composition styles, namely orchestration and choreography, from a modeling perspective. To that end, we have adapted different complexity metrics from the literature to evaluate the complexity of our designed models.

Based on the final results in **Table 10**, we can see that all the complexity measurements used for our evaluation show a higher level of complexity for orchestration. However, in some very rare scenarios both choreography and orchestration have an equal or almost similar level of complexity. Another observation derived from the results of our measurements is that we see an increase in the complexity level as the size of the models gets bigger. Hence, from what we observe, the complexity is almost double in the end-to-end orchestration model compared to the choreography with the same e-commerce scenario. The results of our evaluation support the correlation analysis proposed by Sánchez-González et al. (2010). Based on their study there is a direct connection between the size of business processes and their understandability and modifiability, which they annotate as positive (+) and negative (-) correlations. They use structural metrics, as one of the complexity metrics used in business processes, to measure the size of their proposed business flows. These metrics consider the size of the processes through different measurements including the total number of nodes, diameter, etc. (Sánchez-González et al., 2010, 2017).

Our study uses this correlation analysis to evaluate the understandability and modifiability of the two leading microservice composition approaches, namely orchestration and choreography. The findings of the research show a direct link between the number of nodes used in a BPMN workflow and the complexity. Based on our models, we can see that there are more nodes used in BPMN-based orchestration models compared to choreography with similar scenarios. The size of the workflow directly impacts the level of complexity. In other words, the level of complexity in BPMN models is closely related to the number of services and connectors involved in their process, which also affects the understandability and modifiability of the models (Sánchez-González et al., 2010).

In conclusion, based on our results and considering our modeling technique, choreography reduces the number of nodes and connectors, which results in less complexity compared to orchestration. Therefore, choosing the right composition style is very important for e-commerce applications given the different services involved in their processes. For that e-commerce companies are advised to take multiple factors into consideration: 1. What are the business needs; 2. How many services are involved in the application; 3. what functionality each service performs. Hence, both composition styles come with their limitations that can be overcome by using both choreography and orchestration in the systems, a solution called the hybrid method.

7.2.Conclusion

To conduct this research, we performed a thorough study of the literature to identify the main concepts related to service-oriented architecture, microservice architecture, choreography, orchestration, BPMN modeling in the domain of e-commerce, and the applications of complexity metrics on BPMN choreography and orchestration models. Firstly, we identified the differences between SOA and MSA. Secondly, we uncovered the differences between microservice orchestration and choreography. Thirdly, we measured the level of complexity in BPMN-based choreography and orchestration workflows using complexity metrics from the literature. These complexity metrics helped us get a good understanding of the structure and complexity of microservices choreography and orchestration from a modeling perspective.

Through the findings from the literature, we were able to clarify and uncover several concepts related to microservice compositions and BPMN modeling that we used to answer our research questions announced in Chapter 1 as follows:

- **What are the main differences between choreography and orchestration considering the advantages and disadvantages of each composition style?** Based on the literature and from our findings, both composition techniques have pros and cons, which we have compared in detail in **Table 2**. Therefore, companies should select these composition styles based on their business needs and requirements.

- **Which composition technique (orchestration or choreography) is less complex to deliver business requirements in e-commerce applications based on the proposed scenarios?** The final results from the complexity measurements we applied on our models suggest that orchestration is more complex than choreography for e-commerce applications because there are more services involved in modeling orchestration compared to choreography. We also discuss how complexity can affect the modifiability and understandability of each composition style, which makes choreography models more modifiable and understandable compared to orchestration.

7.3.Research Contributions

This study provides insights into the BPMN modeling of microservice orchestration versus choreography in the domain of e-commerce. The main contribution of this study is to distinguish the differences between choreography and orchestration using complexity as a metric, which provides a better understanding of microservice composition. BPMN modeling techniques and tools allow us to deploy and execute our models to make sure that they are following a correct logic based on real e-commerce processes. While most studies in the literature only propose BPMN modeling without any deployment. Thanks to BPMN 2.0, our models provide a high-level notation of e-commerce workflows using choreography and orchestration. Thus, our models and results can be easily understood by all business users, namely managers, business analysts, and developers.

7.4.Research Strength

Our study is aimed to showcase the composition of microservices on real world e-commerce workflows. The key component of this research is the deployment and execution of our workflows using Zeebe Simple Monitor, which is suggested by Zeebe.io. The tool enables us to test the applicability of our models to the real-world e-commerce workflows without the need to write any code. Another important element of our study is that we have incorporated a communication logic for our models which focuses on synchronous and

asynchronous communication mechanisms of microservices via using conditional sequence flows and XOR gate ways offered by Zeebe Modeler BPMN 2.0 in our workflows.

7.5.Research Limitations and Future Work

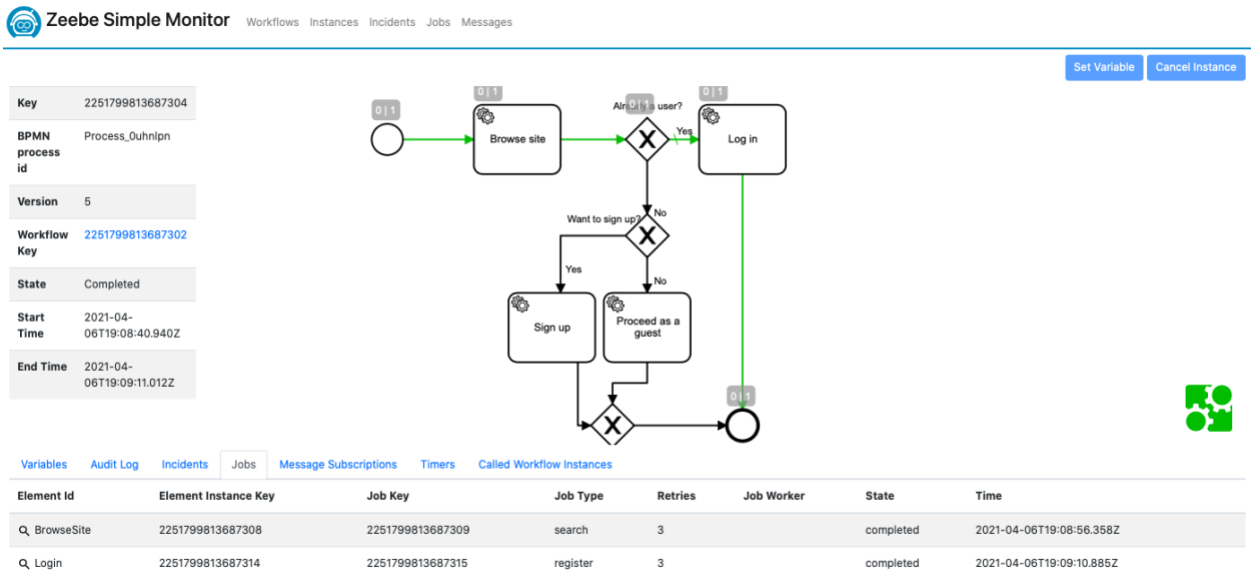
Business processes can be modeled using different modelling techniques and tools. However, for our research we only rely on Zeebe Modeler to develop workflows following the BPMN 2.0 standard, which we consider a limitation of this study. We believe the results can vary depending on what modeling technique and tool is used. The second limitation of this study is that there are some complexity metrics, which we could not use for our measurements, as they require the use of specific BPMN components such as processes and sub-processes. Hence, our use of various complexity metrics is limited to the modeling logic and components used in the workflows. It is important to mention that our research considers complexity as the only metric for comparison, however, based on the literature and as mentioned in Chapter 5, there are other metrics that can be taken into consideration to evaluate and compare microservice compositions.

One other limitation of this research is that all the proposed models are designed based on the scenarios that use in-house service integration instead of third-party services. Therefore, the results can be different when third-party services are integrated in the modeling of the workflows.

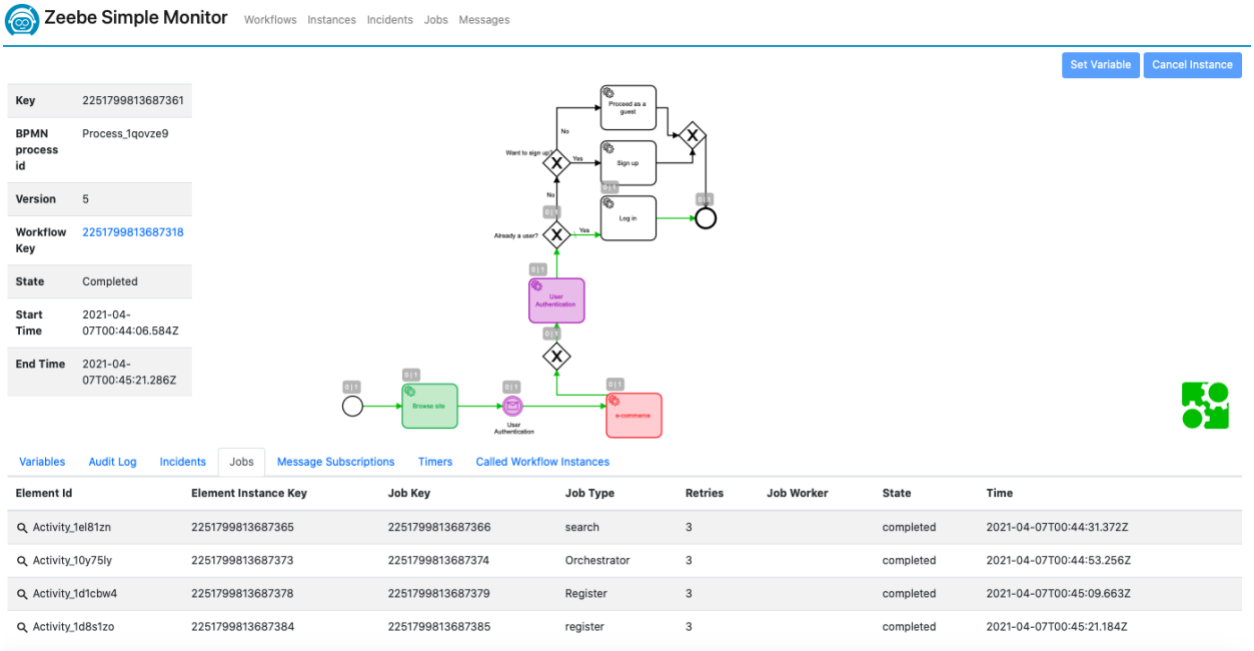
Considering the existing limitations, further research can be done to measure the complexity of microservice compositions using other modeling tools and techniques and compare the results with the existing results to get better insights on the complexity level of choreography and orchestration.

Appendices

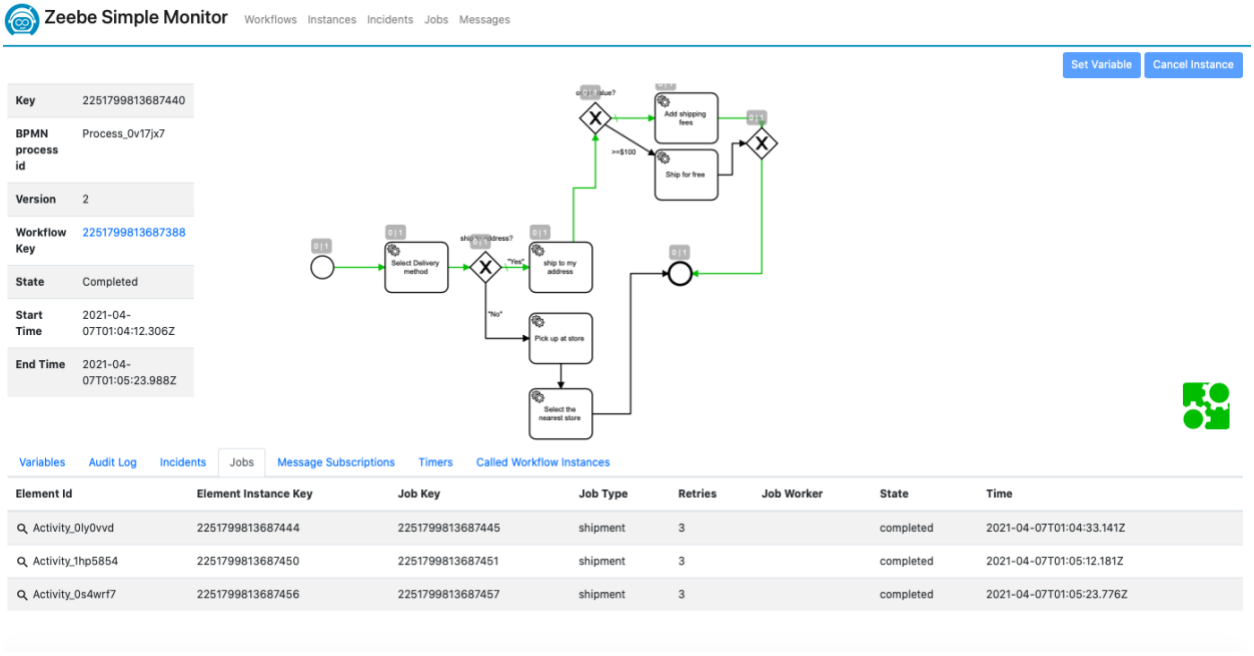
Appendix A. Simulated flow of user authentication choreography on Zeebe Simple Monitor



Appendix B. Simulated flow of user authentication orchestration on Zeebe Simple Monitor

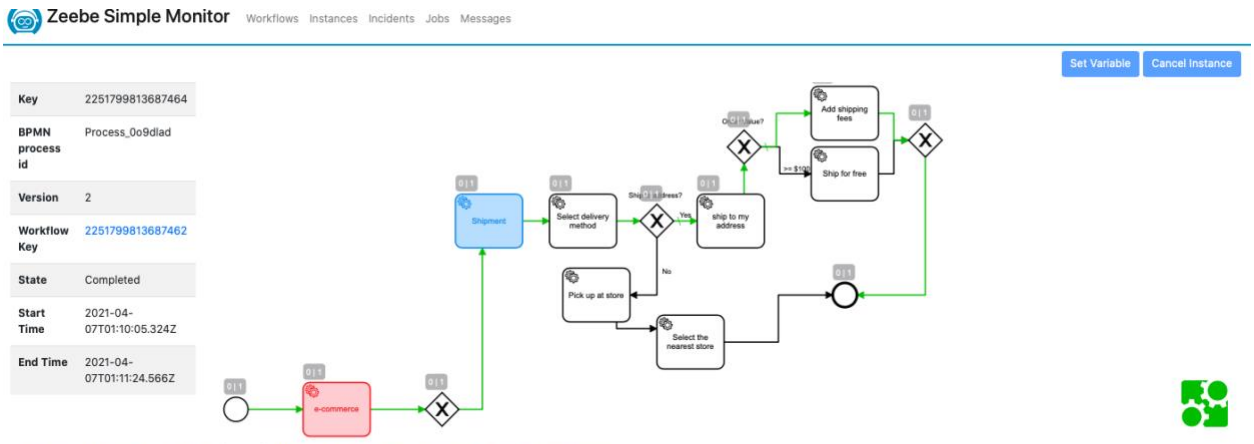


Appendix C. Simulated flow of Shipment choreography on Zeebe Simple Monitor



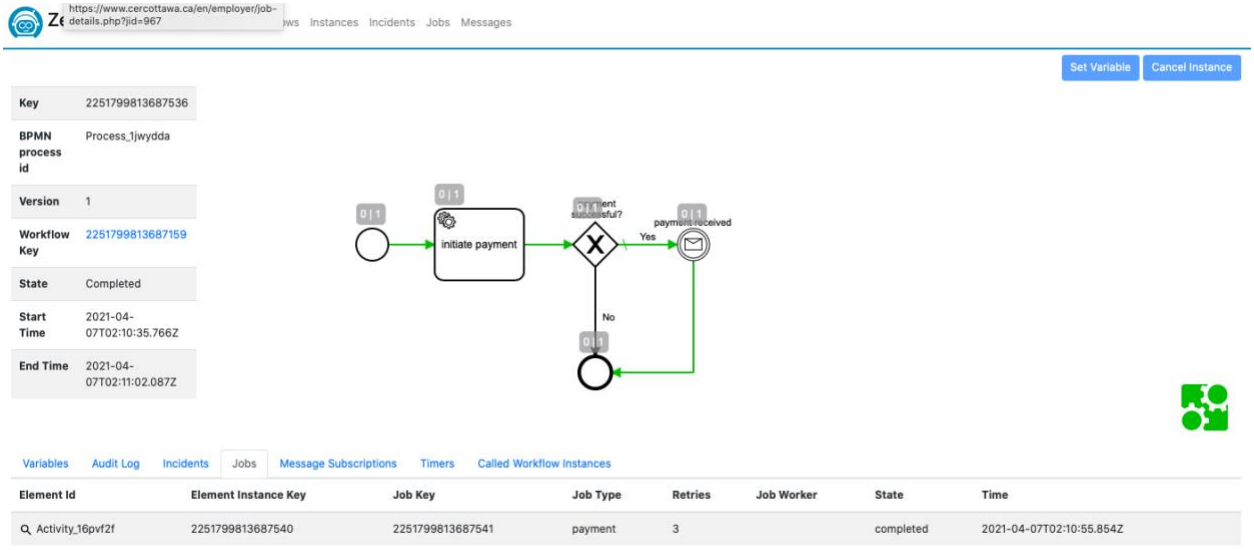
Element Id	Element Instance Key	Job Key	Job Type	Retries	Job Worker	State	Time
Activity_0ly0vvd	2251799813687444	2251799813687445	shipment	3		completed	2021-04-07T01:04:33.141Z
Activity_1hp5854	2251799813687450	2251799813687451	shipment	3		completed	2021-04-07T01:05:12.181Z
Activity_0s4wrf7	2251799813687456	2251799813687457	shipment	3		completed	2021-04-07T01:05:23.776Z

Appendix D. Simulated flow of shipment orchestration on Zeebe Simple Monitor

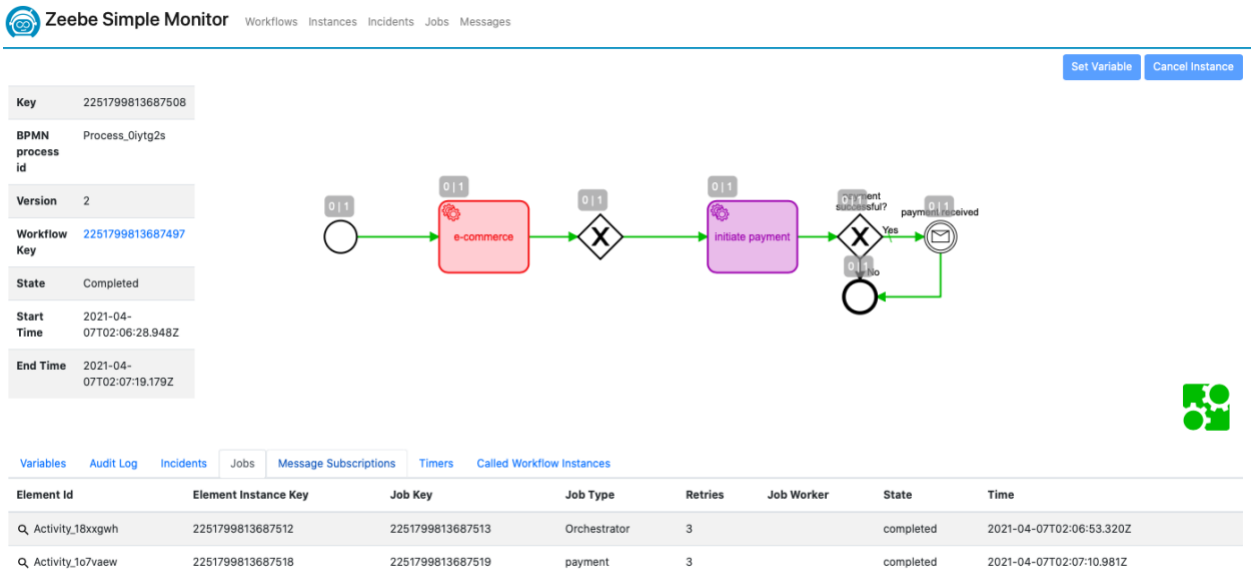


Element Id	Element Instance Key	Job Key	Job Type	Retries	Job Worker	State	Time
Activity_1geh2kc	2251799813687468	2251799813687469	Orchestrator	3		completed	2021-04-07T01:10:25.593Z
Activity_0z206a9	2251799813687474	2251799813687475	shipment	3		completed	2021-04-07T01:10:35.150Z
Activity_0ge0z2l	2251799813687477	2251799813687478	shipment	3		completed	2021-04-07T01:10:53.747Z
Activity_08lmd91	2251799813687483	2251799813687484	shipment	3		completed	2021-04-07T01:11:14.165Z
Activity_109hfr3	2251799813687489	2251799813687490	shipment	3		completed	2021-04-07T01:11:24.406Z

Appendix E. Simulated flow of payment choreography on Zeebe Simple Monitor



Appendix F. Simulated flow of payment orchestration on Zeebe Simple Monitor



Appendix G. XML code for shipment choreography and orchestration

Choreography
<?xml version="1.0" encoding="UTF-8"?> <bpmn:definitions xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"

```
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI" xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:zeebe="http://camunda.org/schema/zeebe/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:di="http://www.omg.org/spec/DD/20100524/DI" id="Definitions_03t337u" targetNamespace="http://bpmn.io/schema/bpmn"
exporter="Zeebe Modeler" exporterVersion="0.9.1">
  <bpmn:process id="Process_0uhnlpn" isExecutable="true">
    <bpmn:startEvent id="StartEvent_1">
      <bpmn:outgoing>Flow_0w9nmdk</bpmn:outgoing>
    </bpmn:startEvent>
    <bpmn:serviceTask id="BrowseSite" name="Browse site">
      <bpmn:extensionElements>
        <zeebe:taskDefinition type="search" />
      </bpmn:extensionElements>
      <bpmn:incoming>Flow_0w9nmdk</bpmn:incoming>
      <bpmn:outgoing>Flow_1kgpqjz</bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:exclusiveGateway id="Gateway_1m2g5pl" name="Already a user?" default="Flow_1p2q14r">
      <bpmn:incoming>Flow_1kgpqjz</bpmn:incoming>
      <bpmn:outgoing>Flow_1p2q14r</bpmn:outgoing>
      <bpmn:outgoing>Flow_0dwhfja</bpmn:outgoing>
    </bpmn:exclusiveGateway>
    <bpmn:serviceTask id="Login" name="Log in">
      <bpmn:extensionElements>
        <zeebe:taskDefinition type="register" />
      </bpmn:extensionElements>
      <bpmn:incoming>Flow_1p2q14r</bpmn:incoming>
      <bpmn:outgoing>Flow_1tddybo</bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:exclusiveGateway id="Gateway_1097d0v" name="Want to sign up?">
      <bpmn:incoming>Flow_0dwhfja</bpmn:incoming>
      <bpmn:outgoing>Flow_1l2vag0</bpmn:outgoing>
      <bpmn:outgoing>Flow_1pz8o7r</bpmn:outgoing>
    </bpmn:exclusiveGateway>
    <bpmn:serviceTask id="Guest" name="Proceed as a guest">
      <bpmn:extensionElements>
        <zeebe:taskDefinition type="register" />
      </bpmn:extensionElements>
      <bpmn:incoming>Flow_1pz8o7r</bpmn:incoming>
      <bpmn:outgoing>Flow_1vifske</bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:serviceTask id="Signup" name="Sign up">
      <bpmn:extensionElements>
        <zeebe:taskDefinition type="register" />
      </bpmn:extensionElements>
      <bpmn:incoming>Flow_1l2vag0</bpmn:incoming>
      <bpmn:outgoing>Flow_1euikrf</bpmn:outgoing>
    </bpmn:serviceTask>
```

```
    <bpmn:exclusiveGateway id="Gateway_0rge0gf">
      <bpmn:incoming>Flow_1vifske</bpmn:incoming>
      <bpmn:incoming>Flow_1euikrf</bpmn:incoming>
      <bpmn:outgoing>Flow_1s299zh</bpmn:outgoing>
    </bpmn:exclusiveGateway>
    <bpmn:sequenceFlow id="Flow_1p2q14r" name="Yes" sourceRef="Gateway_1m2g5pl" targetRef="Login" />
    <bpmn:sequenceFlow id="Flow_0dwhfja" name="No" sourceRef="Gateway_1m2g5pl" targetRef="Gateway_1097d0v">
      <bpmn:conditionExpression xsi:type="bpmn:tFormalExpression">=userId="No"</bpmn:conditionExpression>
    </bpmn:sequenceFlow>
    <bpmn:sequenceFlow id="Flow_1l2vag0" name="Yes" sourceRef="Gateway_1097d0v" targetRef="Signup">
      <bpmn:conditionExpression xsi:type="bpmn:tFormalExpression">=userId="Yes"</bpmn:conditionExpression>
    </bpmn:sequenceFlow>
    <bpmn:sequenceFlow id="Flow_1pz8o7r" name="No" sourceRef="Gateway_1097d0v" targetRef="Guest">
      <bpmn:conditionExpression xsi:type="bpmn:tFormalExpression">=userId="No"</bpmn:conditionExpression>
    </bpmn:sequenceFlow>
    <bpmn:sequenceFlow id="Flow_1vifske" sourceRef="Guest" targetRef="Gateway_0rge0gf" />
    <bpmn:sequenceFlow id="Flow_1euikrf" sourceRef="Signup" targetRef="Gateway_0rge0gf" />
    <bpmn:sequenceFlow id="Flow_0w9nmdk" sourceRef="StartEvent_1" targetRef="BrowseSite" />
    <bpmn:sequenceFlow id="Flow_1kgpqjz" sourceRef="BrowseSite" targetRef="Gateway_1m2g5pl" />
    <bpmn:endEvent id="Event_1kwa4ks">
      <bpmn:incoming>Flow_1s299zh</bpmn:incoming>
      <bpmn:incoming>Flow_1tddybo</bpmn:incoming>
    </bpmn:endEvent>
    <bpmn:sequenceFlow id="Flow_1s299zh" sourceRef="Gateway_0rge0gf" targetRef="Event_1kwa4ks" />
    <bpmn:sequenceFlow id="Flow_1tddybo" sourceRef="Login" targetRef="Event_1kwa4ks" />
  </bpmn:process>
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="Process_0uhnlpn">
      <bpmndi:BPMNEdge id="Flow_1tddybo_di" bpmnElement="Flow_1tddybo">
        <di:waypoint x="580" y="200" />
        <di:waypoint x="580" y="472" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_1s299zh_di" bpmnElement="Flow_1s299zh">
        <di:waypoint x="455" y="490" />
        <di:waypoint x="562" y="490" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_1kgpqjz_di" bpmnElement="Flow_1kgpqjz">
        <di:waypoint x="370" y="160" />
        <di:waypoint x="445" y="160" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_0w9nmdk_di" bpmnElement="Flow_0w9nmdk">
        <di:waypoint x="188" y="160" />
        <di:waypoint x="270" y="160" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_1euikrf_di" bpmnElement="Flow_1euikrf">
        <di:waypoint x="390" y="417" />
```

```
      <di:waypoint x="390" y="490" />
      <di:waypoint x="405" y="490" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="Flow_1vifske_di" bpmnElement="Flow_1vifske">
      <di:waypoint x="470" y="417" />
      <di:waypoint x="470" y="441" />
      <di:waypoint x="430" y="441" />
      <di:waypoint x="430" y="465" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="Flow_1pz8o7r_di" bpmnElement="Flow_1pz8o7r">
      <di:waypoint x="470" y="296" />
      <di:waypoint x="470" y="337" />
      <bpmndi:BPMNLabel>
        <dc:Bounds x="478" y="314" width="15" height="14" />
      </bpmndi:BPMNLabel>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="Flow_1l2vag0_di" bpmnElement="Flow_1l2vag0">
      <di:waypoint x="445" y="271" />
      <di:waypoint x="370" y="271" />
      <di:waypoint x="370" y="337" />
      <bpmndi:BPMNLabel>
        <dc:Bounds x="376" y="301" width="18" height="14" />
      </bpmndi:BPMNLabel>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="Flow_0dwhfja_di" bpmnElement="Flow_0dwhfja">
      <di:waypoint x="470" y="185" />
      <di:waypoint x="470" y="246" />
      <bpmndi:BPMNLabel>
        <dc:Bounds x="478" y="236" width="15" height="14" />
      </bpmndi:BPMNLabel>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="Flow_1p2q14r_di" bpmnElement="Flow_1p2q14r">
      <di:waypoint x="495" y="160" />
      <di:waypoint x="530" y="160" />
      <bpmndi:BPMNLabel>
        <dc:Bounds x="504" y="142" width="18" height="14" />
      </bpmndi:BPMNLabel>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNShape id="_BPMNShape_StartEvent_2" bpmnElement="StartEvent_1">
      <dc:Bounds x="152" y="142" width="36" height="36" />
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="Activity_1xw98ob_di" bpmnElement="BrowseSite">
      <dc:Bounds x="270" y="120" width="100" height="80" />
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="Gateway_1m2g5pl_di" bpmnElement="Gateway_1m2g5pl" isMarkerVisible="true">
      <dc:Bounds x="445" y="135" width="50" height="50" />
```

```
      <bpmndi:BPMNLabel>
        <dc:Bounds x="431" y="113" width="78" height="14" />
      </bpmndi:BPMNLabel>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="Activity_16jsuep_di" bpmnElement="Login">
      <dc:Bounds x="530" y="120" width="100" height="80" />
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="Gateway_1097d0v_di" bpmnElement="Gateway_1097d0v" isMarkerVisible="true">
      <dc:Bounds x="445" y="246" width="50" height="50" />
      <bpmndi:BPMNLabel>
        <dc:Bounds x="378" y="243" width="83" height="14" />
      </bpmndi:BPMNLabel>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="Activity_1f5jxy8_di" bpmnElement="Guest">
      <dc:Bounds x="420" y="337" width="100" height="80" />
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="Activity_0bri7rg_di" bpmnElement="Signup">
      <dc:Bounds x="310" y="337" width="100" height="80" />
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="Gateway_0rge0gf_di" bpmnElement="Gateway_0rge0gf" isMarkerVisible="true">
      <dc:Bounds x="405" y="465" width="50" height="50" />
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="Event_1kwa4ks_di" bpmnElement="Event_1kwa4ks">
      <dc:Bounds x="562" y="472" width="36" height="36" />
    </bpmndi:BPMNShape>
  </bpmndi:BPMNPlane>
 </bpmndi:BPMNDiagram>
</bpmn:definitions>
```

Orchestration

```
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI" xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:zeebe="http://camunda.org/schema/zeebe/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:di="http://www.omg.org/spec/DD/20100524/DI" id="Definitions_03t337u" targetNamespace="http://bpmn.io/schema/bpmn"
exporter="Zeebe Modeler" exporterVersion="0.9.1">
  <bpmn:process id="Process_0uhnlpn" isExecutable="true">
    <bpmn:startEvent id="StartEvent_1">
      <bpmn:outgoing>Flow_0w9nmdk</bpmn:outgoing>
    </bpmn:startEvent>
    <bpmn:serviceTask id="BrowseSite" name="Browse site">
      <bpmn:extensionElements>
        <zeebe:taskDefinition type="search" />
      </bpmn:extensionElements>
      <bpmn:incoming>Flow_0w9nmdk</bpmn:incoming>
```

```
    <bpmn:outgoing>Flow_1kgpqjz</bpmn:outgoing>
  </bpmn:serviceTask>
  <bpmn:exclusiveGateway id="Gateway_1m2g5pl" name="Already a user?" default="Flow_1p2q14r">
    <bpmn:incoming>Flow_1kgpqjz</bpmn:incoming>
    <bpmn:outgoing>Flow_1p2q14r</bpmn:outgoing>
    <bpmn:outgoing>Flow_0dwhfja</bpmn:outgoing>
  </bpmn:exclusiveGateway>
  <bpmn:serviceTask id="Login" name="Log in">
    <bpmn:extensionElements>
      <zeebe:taskDefinition type="register" />
    </bpmn:extensionElements>
    <bpmn:incoming>Flow_1p2q14r</bpmn:incoming>
    <bpmn:outgoing>Flow_1tddybo</bpmn:outgoing>
  </bpmn:serviceTask>
  <bpmn:exclusiveGateway id="Gateway_1097d0v" name="Want to sign up?">
    <bpmn:incoming>Flow_0dwhfja</bpmn:incoming>
    <bpmn:outgoing>Flow_1l2vag0</bpmn:outgoing>
    <bpmn:outgoing>Flow_1pz8o7r</bpmn:outgoing>
  </bpmn:exclusiveGateway>
  <bpmn:serviceTask id="Guest" name="Proceed as a guest">
    <bpmn:extensionElements>
      <zeebe:taskDefinition type="register" />
    </bpmn:extensionElements>
    <bpmn:incoming>Flow_1pz8o7r</bpmn:incoming>
    <bpmn:outgoing>Flow_1vifske</bpmn:outgoing>
  </bpmn:serviceTask>
  <bpmn:serviceTask id="Signup" name="Sign up">
    <bpmn:extensionElements>
      <zeebe:taskDefinition type="register" />
    </bpmn:extensionElements>
    <bpmn:incoming>Flow_1l2vag0</bpmn:incoming>
    <bpmn:outgoing>Flow_1euikrf</bpmn:outgoing>
  </bpmn:serviceTask>
  <bpmn:exclusiveGateway id="Gateway_0rge0gf">
    <bpmn:incoming>Flow_1vifske</bpmn:incoming>
    <bpmn:incoming>Flow_1euikrf</bpmn:incoming>
    <bpmn:outgoing>Flow_1s299zh</bpmn:outgoing>
  </bpmn:exclusiveGateway>
  <bpmn:sequenceFlow id="Flow_1p2q14r" name="Yes" sourceRef="Gateway_1m2g5pl" targetRef="Login" />
  <bpmn:sequenceFlow id="Flow_0dwhfja" name="No" sourceRef="Gateway_1m2g5pl" targetRef="Gateway_1097d0v">
    <bpmn:conditionExpression xsi:type="bpmn:tFormalExpression">=userId="No"</bpmn:conditionExpression>
  </bpmn:sequenceFlow>
  <bpmn:sequenceFlow id="Flow_1l2vag0" name="Yes" sourceRef="Gateway_1097d0v" targetRef="Signup">
    <bpmn:conditionExpression xsi:type="bpmn:tFormalExpression">=userId="Yes"</bpmn:conditionExpression>
  </bpmn:sequenceFlow>
  <bpmn:sequenceFlow id="Flow_1pz8o7r" name="No" sourceRef="Gateway_1097d0v" targetRef="Guest">
```

```
    <bpmn:conditionExpression xsi:type="bpmn:tFormalExpression">=userId="No"</bpmn:conditionExpression>
  </bpmn:sequenceFlow>
  <bpmn:sequenceFlow id="Flow_1vifske" sourceRef="Guest" targetRef="Gateway_0rge0gf" />
  <bpmn:sequenceFlow id="Flow_1euikrf" sourceRef="Signup" targetRef="Gateway_0rge0gf" />
  <bpmn:sequenceFlow id="Flow_0w9nmdk" sourceRef="StartEvent_1" targetRef="BrowseSite" />
  <bpmn:sequenceFlow id="Flow_1kgpqjz" sourceRef="BrowseSite" targetRef="Gateway_1m2g5pl" />
  <bpmn:endEvent id="Event_1kwa4ks">
    <bpmn:incoming>Flow_1s299zh</bpmn:incoming>
    <bpmn:incoming>Flow_1tddybo</bpmn:incoming>
  </bpmn:endEvent>
  <bpmn:sequenceFlow id="Flow_1s299zh" sourceRef="Gateway_0rge0gf" targetRef="Event_1kwa4ks" />
  <bpmn:sequenceFlow id="Flow_1tddybo" sourceRef="Login" targetRef="Event_1kwa4ks" />
</bpmn:process>
<bpmndi:BPMNDiagram id="BPMNDiagram_1">
  <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="Process_0uhnlpn">
    <bpmndi:BPMNEdge id="Flow_1tddybo_di" bpmnElement="Flow_1tddybo">
      <di:waypoint x="580" y="200" />
      <di:waypoint x="580" y="472" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="Flow_1s299zh_di" bpmnElement="Flow_1s299zh">
      <di:waypoint x="455" y="490" />
      <di:waypoint x="562" y="490" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="Flow_1kgpqjz_di" bpmnElement="Flow_1kgpqjz">
      <di:waypoint x="370" y="160" />
      <di:waypoint x="445" y="160" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="Flow_0w9nmdk_di" bpmnElement="Flow_0w9nmdk">
      <di:waypoint x="188" y="160" />
      <di:waypoint x="270" y="160" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="Flow_1euikrf_di" bpmnElement="Flow_1euikrf">
      <di:waypoint x="390" y="417" />
      <di:waypoint x="390" y="490" />
      <di:waypoint x="405" y="490" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="Flow_1vifske_di" bpmnElement="Flow_1vifske">
      <di:waypoint x="470" y="417" />
      <di:waypoint x="470" y="441" />
      <di:waypoint x="430" y="441" />
      <di:waypoint x="430" y="465" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="Flow_1pz8o7r_di" bpmnElement="Flow_1pz8o7r">
      <di:waypoint x="470" y="296" />
      <di:waypoint x="470" y="337" />
      <bpmndi:BPMNLabel>
```

```
      <dc:Bounds x="478" y="314" width="15" height="14" />
    </bpmndi:BPMNLabel>
  </bpmndi:BPMNEdge>
  <bpmndi:BPMNEdge id="Flow_1l2vag0_di" bpmnElement="Flow_1l2vag0">
    <di:waypoint x="445" y="271" />
    <di:waypoint x="370" y="271" />
    <di:waypoint x="370" y="337" />
    <bpmndi:BPMNLabel>
      <dc:Bounds x="376" y="301" width="18" height="14" />
    </bpmndi:BPMNLabel>
  </bpmndi:BPMNEdge>
  <bpmndi:BPMNEdge id="Flow_0dwhfja_di" bpmnElement="Flow_0dwhfja">
    <di:waypoint x="470" y="185" />
    <di:waypoint x="470" y="246" />
    <bpmndi:BPMNLabel>
      <dc:Bounds x="478" y="236" width="15" height="14" />
    </bpmndi:BPMNLabel>
  </bpmndi:BPMNEdge>
  <bpmndi:BPMNEdge id="Flow_1p2q14r_di" bpmnElement="Flow_1p2q14r">
    <di:waypoint x="495" y="160" />
    <di:waypoint x="530" y="160" />
    <bpmndi:BPMNLabel>
      <dc:Bounds x="504" y="142" width="18" height="14" />
    </bpmndi:BPMNLabel>
  </bpmndi:BPMNEdge>
  <bpmndi:BPMNShape id="_BPMNShape_StartEvent_2" bpmnElement="StartEvent_1">
    <dc:Bounds x="152" y="142" width="36" height="36" />
  </bpmndi:BPMNShape>
  <bpmndi:BPMNShape id="Activity_1xw98ob_di" bpmnElement="BrowseSite">
    <dc:Bounds x="270" y="120" width="100" height="80" />
  </bpmndi:BPMNShape>
  <bpmndi:BPMNShape id="Gateway_1m2g5pl_di" bpmnElement="Gateway_1m2g5pl" isMarkerVisible="true">
    <dc:Bounds x="445" y="135" width="50" height="50" />
    <bpmndi:BPMNLabel>
      <dc:Bounds x="431" y="113" width="78" height="14" />
    </bpmndi:BPMNLabel>
  </bpmndi:BPMNShape>
  <bpmndi:BPMNShape id="Activity_16jsuep_di" bpmnElement="Login">
    <dc:Bounds x="530" y="120" width="100" height="80" />
  </bpmndi:BPMNShape>
  <bpmndi:BPMNShape id="Gateway_1097d0v_di" bpmnElement="Gateway_1097d0v" isMarkerVisible="true">
    <dc:Bounds x="445" y="246" width="50" height="50" />
    <bpmndi:BPMNLabel>
      <dc:Bounds x="378" y="243" width="83" height="14" />
    </bpmndi:BPMNLabel>
  </bpmndi:BPMNShape>
```

```
      <bpmndi:BPMNShape id="Activity_1f5jxy8_di" bpmnElement="Guest">
        <dc:Bounds x="420" y="337" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Activity_0bri7rg_di" bpmnElement="Signup">
        <dc:Bounds x="310" y="337" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Gateway_0rge0gf_di" bpmnElement="Gateway_0rge0gf" isMarkerVisible="true">
        <dc:Bounds x="405" y="465" width="50" height="50" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Event_1kwa4ks_di" bpmnElement="Event_1kwa4ks">
        <dc:Bounds x="562" y="472" width="36" height="36" />
      </bpmndi:BPMNShape>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</bpmn:definitions>
```

References

Asrowardi, I., Putra, S. D., & Subyantoro, E. (2020). Designing microservice architectures for scalability and reliability in e-commerce. *Journal of Physics: Conference Series*, *1450*, 012077. https://doi.org/10.1088/1742-6596/1450/1/012077

Baboi, M., Iftene, A., & Gîfu, D. (2019). Dynamic microservices to create scalable and fault tolerance architecture. *Procedia Computer Science*, *159*, 1035–1044. https://doi.org/10.1016/j.procs.2019.09.271

Balalaie, A., Heydarnoori, A., Jamshidi, P., Tamburri, D. A., & Lynn, T. (2018). Microservices migration patterns. *Software - Practice and Experience*, *48*(11), 2019–2042. https://doi.org/10.1002/spe.2608

Banerjee, S. (2018). *Concepts and Tools for Measuring the Complexity of Service Choreography Models*.

Baškarada, S., Nguyen, V., & Koronios, A. (2018). Architecting Microservices: Practical Opportunities and Challenges. *Journal of Computer Information Systems*, *00*(00), 1–9. https://doi.org/10.1080/08874417.2018.1520056

Baumeister, H. (2019). Designing for Optimum Experiences in Online Retail: The Impact of Website Design on Flow in Online Retail Environments. *Indo American Journal of Pharmaceutical Sciences*, *23*(3), 6. https://doi.org/10.5281/zenodo.1477753

Belanger, F., Fan, W., Schaupp, L. C., Krishen, A., Everhart, J., Poteet, D., & Nakamoto, K. (2006). Web site success metrics. *Communications of the ACM*, *49*(12), 114. https://doi.org/10.1145/1183236.1183256

Bhatti, A., Akram, H., Basit, H. M., Khan, A. U., & Raza, S. M. (2020). *E-commerce trends during COVID-19 Pandemic*. *13*(2), 1449–1452.

Bigheti, J. A., Fernandes, M. M., & Godoy, E. D. P. (2019). Control as a Service: A Microservice Approach to Industry 4.0. *2019 IEEE International Workshop on Metrology for Industry 4.0 and IoT, MetroInd 4.0 and IoT 2019 - Proceedings*, 438–443. https://doi.org/10.1109/METROI4.2019.8792918

Bocciarelli, P., Pieroni, A., Gianni, D., & D'Ambrogio, A. (2012). A model-driven method for building distributed simulation systems from business process models. *Proceedings - Winter Simulation Conference*. https://doi.org/10.1109/WSC.2012.6465106

Bogner, J., Wagner, S., & Zimmermann, A. (2019). Using architectural modifiability tactics to examine evolution qualities of Service- and Microservice-Based Systems: An approach based on principles and patterns. *Software-Intensive Cyber-Physical Systems*, *34*(2–3), 141–149. https://doi.org/10.1007/s00450-019-00402-z

Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S. T., & Mazzara, M. (2018). From Monolithic to Microservices: An Experience Report from the Banking Domain. *IEEE Software*, *35*(3), 50–55. https://doi.org/10.1109/MS.2018.2141026

Butzin, B., Golatowski, F., & Timmermann, D. (2016). Microservices approach for the internet of things. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, *2016-Novem*. https://doi.org/10.1109/ETFA.2016.7733707

Camilli, M., Bellettini, C., Capra, L., & Monga, M. (2018). A formal framfile:///home/per/Downloads/10.1.1.212.7429.pdf ework for specifying and verifying microservices based process flows. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10729 LNCS*, 187–202. https://doi.org/10.1007/978-3-319-74781-1_14

Cardoso, J, Mendling, J., Neumann, G., & Reijers, H. A. (2006). A Discourse on Complexity of Process Models (Survey Paper). *BPM 2006 Workshops, Lecture Notes in Computer Science 4103*, 117–128.

Cardoso, Jorge. (2005). Control-flow Complexity Measurement of Processes and Weyuker's Properties. *6th International Conference on Enformatika*, *8*, 213–218.

Cer, T., & Donahoo, M. J. (2017). *Contextual Understanding of Microservice Architecture : Current and Future Directions*. 1–20.

Cerny, T., Donahoo, M. J., & Trnka, M. (2018). Contextual understanding of microservice architecture. *ACM SIGAPP Applied Computing Review*, *17*(4), 29–45. https://doi.org/10.1145/3183628.3183631

Chu, S. C., Leung, L. C., Hui, Y. Van, & Cheung, W. (2007). Evolution of e-commerce Web sites: A conceptual framework and a longitudinal study. *Information and Management*, *44*(2), 154–164. https://doi.org/10.1016/j.im.2006.11.003

Di Francesco, P., Malavolta, I., & Lago, P. (2017). Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. *Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017*, 21–30.

https://doi.org/10.1109/ICSA.2017.24

Gutiérrez–Fernández, A. M., Resinas, M., & Ruiz–Cortés, A. (2017). Redefining a process engine as a microservice platform. *Lecture Notes in Business Information Processing*, *281*, 252–263. https://doi.org/10.1007/978-3-319-58457-7_19

HALSTEAD, M. H. (1977). Elementsof software sct- ence, Elsevier North-Holland, Inc, N Y. *ACM Computing Surveys (CSUR)*, *10*(1), 3–18. https://doi.org/10.1145/356715.356717

Hasselbring, W., & Steinacker, G. (2017). Microservice architectures for scalability, agility and reliability in e-commerce. *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*, 243–246. https://doi.org/10.1109/ICSAW.2017.11

Henry & Kafura. (1974). On information flow in ordination. *Vegetatio*, *29*(1), 11–16. https://doi.org/10.1007/BF02390891

Hevner, A. R., March, S. T., Park, J., Ram, S., & Ram, S. (2004). Research Essay Design Science in Information. *MIS Quarterly*, *28*(1), 75–105. https://doi.org/10.2307/25148625

Isoyama, K., Kobayashi, Y., Sato, T., Kida, K., Yoshida, M., & Tagato, H. (2012). Short Paper: A scalable complex event processing system and evaluations of its performance. *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, DEBS'12*, 123–126. https://doi.org/10.1145/2335484.2335498

Kaimann. (1974). *Coefficient of Network Complexity Author (s): Richard A . Kaimann Source : Management Science , Oct ., 1974 , Vol . 21 , No . 2 , Application Series (Oct ., 1974), Published by : INFORMS Stable URL : https://www.jstor.org/stable/2629677 COEFFICIENT OF*. *21*(2), 172–177.

Kluza, K., & Nalepa, G. J. (2012). Proposal of square metrics for measuring Business Process Model complexity. *2012 Federated Conference on Computer Science and Information Systems, FedCSIS 2012*, 919–922.

Kluza, K., Nalepa, G. J., & Lisiecki, J. (2014). Square complexity metrics for business process models. *Advances in Intelligent Systems and Computing*, *257*(October), 89–107. https://doi.org/10.1007/978-3-319-03677-9_6

Kouchaksaraei, H. R., Dierich, T., & Karl, H. (2018). Pishahang: Joint Orchestration of Network Function Chains and Distributed Cloud Applications. *2018 4th IEEE Conference on Network Softwarization and Workshops, NetSoft 2018*, *NetSoft*, 308–312.

https://doi.org/10.1109/NETSOFT.2018.8460134

Latva-Koivisto, A. M. (2001). Finding a complexity measure for business process models. *Complexity*, 1–26. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.2991&rep=rep1&type=pdf

López-Miguens, M. J., & Vázquez, E. G. (2017). An integral model of e-loyalty from the consumer's perspective. *Computers in Human Behavior*, *72*, 397–411. https://doi.org/10.1016/j.chb.2017.02.003

Mazzara, M., Dragoni, N., Bucchiarone, A., Giaretta, A., Larsen, S. T., & Dustdar, S. (2017). Microservices: Migration of a Mission Critical System. *IEEE Transactions on Services Computing*. https://doi.org/10.1109/TSC.2018.2889087

Mccabe, T. J. (1976). *A Complexi-ty*. *4*, 308–320.

Monsalve, C., Abran, A., & April, A. (2011). Measuring software functional size from business process models. *International Journal of Software Engineering and Knowledge Engineering*, *21*(3), 311–338. https://doi.org/10.1142/S0218194011005359

Monteiro, D., Gadelha, R., Maia, P. H. M., Rocha, L. S., & Mendonça, N. C. (2018). Beethoven: An Event-Driven Lightweight Platform for Microservice Orchestration Davi. *European Conference on Software Architecture (Pp. 191-199). Springer, Cham.*, *1*(September), 191–199. https://doi.org/10.1007/978-3-030-00761-4

Nehme, A., Jesus, V., Mahbub, K., & Abdallah, A. (2019). Securing Microservices. *IT Professional*, *21*(1), 42–49. https://doi.org/10.1109/MITP.2018.2876987

Newman, S. (2015). Building Microservices. In *O'Reilly*. https://www.google.hr/books?hl=en&lr=&id=jjl4BgAAQBAJ&pgis=1%5Cnhttp://oreilly.com/catalog/errata.csp?isbn=9781491950357

Nkomo, P., & Coetzee, M. (2019). *Software Development Activities for secure Microservices*. In International Conference on Computational Science and Its Applications (pp. 573-585). Springer, Cham. https://doi.org/10.1007/978-3-030-24308-1

Oberhauser, R., & Stigler, S. (2017). Microflows: Enabling agile business process modeling to orchestrate semantically-annotated microservices. *BMSD 2017 - Proceedings of the 7th International Symposium on Business Modeling and Software Design*, *Bmsd*, 19–28. https://doi.org/10.5220/0006527100190028

Ortin, F., & O'Shea, D. (2018). Towards an Easily Programmable IoT Framework Based on Microservices. *Journal of Software*, *13*(1), 90–102. https://doi.org/10.17706/jsw.13.2.90-102

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Peltz, C. (2003). Web services orchestration and choreography. *Computer*, *36*(10), 46–52. https://doi.org/10.1109/MC.2003.1236471

Rolón, E., Ruiz, F., García, F., & Piattini, M. (2006). Applying Software Metrics to evaluate Business Process Models. *CLEI Electronic Journal*, *9*(1). https://doi.org/10.19153/cleiej.9.1.5

Rudrabhatla, C. K. (2018a). Comparison of event choreography and orchestration techniques in Microservice Architecture. *International Journal of Advanced Computer Science and Applications*, *9*(8), 18–22. https://doi.org/10.14569/ijacsa.2018.090804

Rudrabhatla, C. K. (2018b). Comparison of event choreography and orchestration techniques in Microservice Architecture. *International Journal of Advanced Computer Science and Applications*, *9*(8), 18–22. https://doi.org/10.14569/ijacsa.2018.090804

Safavi, R. (2009). Interface design issues to enhance usability of E-commerce websites and systems. *ICCTD 2009 - 2009 International Conference on Computer Technology and Development*, *1*(2000), 277–281. https://doi.org/10.1109/ICCTD.2009.23

Sánchez-González, L., García, F., Mendling, J., Ruiz, F., & Piattini, M. (2010). Prediction of business process model quality based on structural metrics. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6412 LNCS*, 458–463. https://doi.org/10.1007/978-3-642-16373-9_35

Sánchez-González, L., García, F., Ruiz, F., & Mendling, J. (2012). Quality indicators for business process models from a gateway complexity perspective. *Information and Software Technology*, *54*(11), 1159–1174. https://doi.org/10.1016/j.infsof.2012.05.001

Sánchez-González, L., García, F., Ruiz, F., & Piattini, M. (2017). A case study about the improvement of business process models driven by indicators. *Software and Systems Modeling*, *16*(3), 759–788. https://doi.org/10.1007/s10270-015-0482-0

Scattone, F. F., & Braghetto, K. R. (2019). A microservices architecture for distributed Complex

Event Processing in smart cities. *Proceedings - 2018 IEEE 37th International Symposium on Reliable Distributed Systems Workshops, SRDSW 2018*, 6–9. https://doi.org/10.1109/SRDSW.2018.00012

Shadija, D., Rezai, M., & Hill, R. (2017). *Towards an Understanding of Microservices Dharmendra*. *September*. https://doi.org/10.1037/h0040968

Shrestha, H., & Vuorimaa, P. (2019). *Design Science Methodology for Microservice Architecture and System Research*.

Singhal, N., Sakthivel, U., & Raj, P. (2019a). Efficient hybrid research for QoS-aware microservice composition. *International Journal of Recent Technology and Engineering*, *8*(2), 5251–5255. https://doi.org/10.35940/ijrte.B1055.078219

Singhal, N., Sakthivel, U., & Raj, P. (2019b). Selection Mechanism of Micro-Services Orchestration Vs. Choreography. *International Journal of Web & Semantic Technology*, *10*(1), 01–13. https://doi.org/10.5121/ijwest.2019.10101

Solichah, I., Hamilton, M., Mursanto, P., Ryan, C., & Perepletchikov, M. (2013). Exploration on software complexity metrics for business process model and notation. *2013 International Conference on Advanced Computer Science and Information Systems, ICACSIS 2013*, 31–37. https://doi.org/10.1109/ICACSIS.2013.6761549

Systems, S., Cerny, T., & Donahoo, M. J. (n.d.). *Disambiguation and Comparison of*. 228–235. https://doi.org/10.2337/dc07-2424

Thramboulidis, K., Vachtsevanou, D. C., & Solanos, A. (2018). Cyber-physical microservices: An IoT-based framework for manufacturing systems. *Proceedings - 2018 IEEE Industrial Cyber-Physical Systems, ICPS 2018*, 232–239. https://doi.org/10.1109/ICPHYS.2018.8387665

Valderas, P. (2020). Supporting a Hybrid Composition of Microservices. The EUCalipTool Platform. *Journal of Software Engineering Research and Development*, *8*, 1. https://doi.org/10.5753/jserd.2020.457

Valderas, P., Torres, V., & Pelechano, V. (2020). A microservice composition approach based on the choreography of BPMN fragments. *Information and Software Technology*, *127*(July 2019). https://doi.org/10.1016/j.infsof.2020.106370

Vanderfeesten, I., Cardoso, J., Reijers, H. A., & Aalst, W. Van Der. (2007). *Quality Metrics for Business Process Models*. *August 2015*. https://doi.org/10.1007/978-3-540-89224-3

Venable, J. R., Pries-heje, J., & Baskerville, R. (2017). Choosing a Design Science Research Methodology. *Australasian Conference on Information Systems*, 1–11. https://www.acis2017.org/wp-content/uploads/2017/11/ACIS2017_paper_255_FULL.pdf

Viennot, N., Lécuyer, M., Bell, J., Geambasu, R., & Nieh, J. (2015). Synapse: A microservices architecture for heterogeneous-database web applications. *Proceedings of the 10th European Conference on Computer Systems, EuroSys 2015*. https://doi.org/10.1145/2741948.2741975

Wang, H., Khoshgoftaar, T. M., Van Hulse, J., & Gao, K. (2011). Metric selection for software defect prediction. *International Journal of Software Engineering and Knowledge Engineering, 21*(2), 237–257. https://doi.org/10.1142/S0218194011005256

Xiao, Z., Wijegunaratne, I., & Qiang, X. (2017). Reflections on SOA and Microservices. *Proceedings - 4th International Conference on Enterprise Systems: Advances in Enterprise Systems, ES 2016*, 60–67. https://doi.org/10.1109/ES.2016.14

Yahia, E. B. H., Réveillère, L., Bromberg, Y. D., Chevalier, R., & Cadot, A. (2016). Medley: An event-driven lightweight platform for service composition. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9671*, 3–20. https://doi.org/10.1007/978-3-319-38791-8_1

Zeebe.io. (n.d.). *Introduction - Zeebe Documentation*.

Zimmermann, O. (2016). Microservices Tenets: Agile Approach to Service Development and Deployment Overview and Vision Paper, SummerSoC 2016. *Computer Science-Research and Development, 32*(3), 301–310. https://doi.org/10.1007/s00450-016-0337-0