Fall 10-17-2022

Measuring the modeling complexity of microservice choreography and orchestration: The case of e-commerce applications

Mahtab Haj Ali
University of Ottawa, Ottawa, Ontario, Canada, mhaja008@uottawa.ca

Razibul Hasan
University of Ottawa, Ottawa, Ontario, Canada, rhasa044@uottawa.ca

Morad Benyoucef
University of Ottawa, Ottawa, Ontario, Canada, benyoucef@telfer.uottawa.ca

Haj Ali, M., Hasan, R. & Benyoucef, M. (2022). Measuring the modeling complexity of microservice choreography and orchestration: The case of e-commerce applications. In Li, E.Y. *et al.* (Eds.) *Proceedings of The International Conference on Electronic Business, Volume 22* (pp. 299-314). ICEB'22, Bangkok, Thailand, October 13-17, 2022

Measuring the modeling complexity of microservice choreography and orchestration: The case of e-commerce applications

Mahtab Haj Ali [1,*]
Razibul Hasan [2]
Morad Benyoucef [3]

*Corresponding author
[1] Master's Graduate, University of Ottawa, Ottawa, Canada, mhaja008@uottawa.ca
[2] Master's Student, University of Ottawa, Ottawa, Canada, rhasa044@uottawa.ca
[3] Full Professor, University of Ottawa, benyoucef@telfer.uottawa.ca

ABSTRACT

Context: 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 needs. Hence, this research uses existing complexity metrics from the software engineering and business process modeling domains on small, mid-sized, and end-to-end e-commerce scenarios to analyze and compare the level of complexity of microservice Orchestration and Choreography using Business Process Modeling Notation (BPMN).

Objective: Comparing 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.

Method: This research follows a five-step process for conducting a Design Science Research (DSR) methodology to define, develop and evaluate BPMN-based models for microservice compositions.

Results: A series of BPMN workflows are designed as artifacts to investigate microservice Choreography and Orchestration. The results derived from the complexity evaluation of our proposed models show a higher level of complexity in orchestrating microservices for e-commerce applications given the number of services used in modeling Orchestration compared to Choreography.

Conclusion: This research uncovers insights on modeling microservice Choreography and Orchestration and discusses the impacts of complexity on the modifiability and understandability of the proposed models.
Keywords: Microservice, Microservice Composition, Choreography, Orchestration, Complexity Metric, BPMN.

Keywords: Microservice, Microservice Composition, Choreography, Orchestration, Complexity Metric, BPMN.

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's 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 is a complex undertaking in monolithic applications (Newman, 2015). The 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 a 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 perform 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 based applications are composed of multiple microservices, it is possible to use different technologies to meet the requirements of each microservice and avoid choosing one standardized technology, which increases the robustness of the code (Nehme et al., 2019).

However, these microservices need to collaborate with each other to complete their tasks and achieve the outcome of 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. Some recent studies have suggested using a combination of Choreography and Orchestration which is called a hybrid composition method. 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.

In this paper, we assess the complexity of the two leading microservice composition techniques. Complexity has been used in the literature as a metric to evaluate business process models (Solichah et al., 2013),(Kluza et al., 2014; Kluza & Nalepa, 2012), (Haouari & Ghannouchi, 2017; Rolón et al., 2009). We use this metric to assess the complexity of BPMN models for the Choreography and Orchestration of microservices. These metrics are numerical expressions of the models' complexity and structure which are derived from software engineering performance metrics. They embody a quantitative measurement of the maintainability of the models as well as the ease of forecasting errors in such models (Banerjee, 2018). These metrics can help in measuring the complexity level in microservice Orchestration and Choreography to assess the models' difficulty level with regards to their understandability and maintainability.

Problem Statement

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 answer the following research question:

> Which composition technique (Orchestration or Choreography) leads to less complex models to deliver the business requirements in e-commerce applications?

Main Contributions

This study provides insights into the BPMN modeling of microservice Orchestration and Choreography in the domain of e-commerce applications. The main contribution of this study is to distinguish the differences between Choreography and Orchestration using complexity metrics, 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 scenarios. Note that most studies in the literature only propose BPMN modeling without any deployment or execution. Thanks to BPMN 2.0, our models provide a high-level notation of e-commerce workflows using Choreography and Orchestration, hence they can be easily understood by all stakeholders, namely managers, business analysts, and developers.

Research Methodology

We rely on the six-step process of Design Science Research (DSR) (Peffers et al., 2007). The design-science paradigm is basically aimed 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).

The six-step process of the DSR methodology includes: (1) identify the research problem and motivation, (2) define the objective of a solution, (3) design and development, (4) demonstration, (5) evaluation, and (6) communication. Following the abovementioned steps, we first 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 and real-world e-commerce scenarios in the industry. 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. 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 a Choreography and an Orchestration for each scenario. In Step 4, we use Zeebe BPMN Modeler, Zeebe-docker, CAMUNDA automation engine (Zeebe.io, 2021), and Amazon Web Services-AWS cloud to design (i.e., model the workflows or microservice compositions) and deploy the microservices on the cloud. Once we have the workflows designed and deployed, we evaluate them 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 to assess and compare our models and draw insights. Finally, 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.

Structure of the Paper

The remainder of this paper is structured as follows. Section 2 features the background and related work about microservice composition techniques and state of the art in complexity metrics. Section 3 presents complexity measurement approaches for business processes. Section 4 outlines the logic we proposed to model and execute microservice compositions. Section 5 focuses on the implementation and evaluation. Section 5 focuses on the results we obtain from the evaluation of our models. Finally, we use the results of the evaluation to draw conclusions and propose suggestions for future work in this area.

BACKGROUND

Research Domain

Our research domain is e-commerce applications, and there are two main reasons for that. 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, market share, and customer loyalty. E-commerce applications have increased significantly as a way for companies to promote their business (Asrowardi et al., 2020). 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, e-commerce being one of the enablers of such transformation (Bhatti et al., 2020). However, as e-commerce grows there is a need for more advanced technologies such as microservices to support and give more flexibility to online shopping platforms. Given the various services involved in e-commerce applications, microservices can potentially improve such applications by offering loosely coupled microservices allowing them to deliver services to users from anywhere, anytime in an uninterrupted and dynamic fashion. Microservice concepts and tools can make a significant transformation within the e-commerce industry to allow hundreds of modules to work in parallel. Hasselbring & Steinacker (2017), for instance, address the use of microservices in one of the biggest European e-commerce platforms called Otto.de. They discuss the importance of microservice technology in the domain of e-commerce and how microservices improve scalability, reliability, and agility of the Otto.de website by using a vertical structure. Their study shows another useful feature of a microservice architecture which makes them well suited for e-commerce applications, namely high consistency. This 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. Other important features of microservice composition are fault tolerance and resiliency of the system. Given the cross-functional design of microservices, each microservice works independently and failing of one microservice will not affect the entire system (Hasselbring & Steinacker, 2017).

Given the applicability and potential of microservices in the domain of e-commerce, our study is aimed 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 provide 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.

Related Work

Microservice composition shows service collaborations, the business process, and the sequence of the activities in one application. In other words, it is used to deploy and coordinate services in a business application. As mentioned earlier, there are two well-known composition techniques that are used in microservice applications, Choreography and Orchestration. Below we provide a brief overview of each composition technique.

Orchestration

In Orchestration, all microservice interactions are controlled by a central controller that functions similar to an orchestrator (Rudrabhatla 2018). The Orchestrator is responsible for the entire communication in the system. The central controller manages all the requests and service calls. This centralized environment uses request/response messages as a communication mechanism. The central controller calls a service by sending a request to that microservice and waits for it to respond before sending a request to the next microservice (Rudrabhatla, 2018). The next microservice cannot be called until the called microservice sends the proper response to the Orchestrator. This increases the waiting time and dependency between microservices (Valderas, 2020).

Choreography

In Choreography, there is no central controller, so microservices work independently. The output of a microservice is the input of another microservice. This approach uses an event-driven architecture pattern for microservices which makes this approach relatively complex compared to Orchestration (Baškarada et al., 2018; Isoyama et al., 2012; Rudrabhatla, 2018). 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 (Cerny et al., 2018, 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".

A Comparison of Choreography and Orchestration

Microservice composition captures service collaborations, the business process, and the sequence of activities in one application. In other words, it is mainly used to deploy and coordinate services in an application. In this subsection, we use the literature to evaluate the advantages and disadvantages of microservice choreography and orchestration (see Table 1).

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. The process ends when there are no more events to be broadcasted (Isoyama et al., 2012; Kluza & Nalepa, 2012). There is no central controller in this composition technique to listen to the transactions and call the right microservice to execute the transaction.

The other composition style is orchestration in which there is a coordinator that listens to the events published by any of the microservices' 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 the 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 (Kluza & Nalepa, 2012).

Table 1: Choreography Vs. Orchestration

Features	Orchestration	Choreography	References
Monitoring	✓ Easier thanks to the central conductor	✓ No monitoring, as microservices are responsible for monitoring their performance	(Cerny et al., 2018)
Error fixing	✓ Easier to detect errors as all the tasks are constantly monitored by the orchestrator	✓ Hard to detect errors but errors cannot affect the entire system	(Nkomo & Coetzee, 2019)
Scalability	✓ Offers low scalability as it is hard to add a new service	✓ Offers a high level of scalability since all services work independently and a new service can be added more easily	(Cerny et al., 2018)
Speed	✓ More latency due to send/request communication technique	✓ No or very little latency due to event-based communication	(Kluza et al., 2014; Kluza & Nalepa, 2012)
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 is happening (i.e., the inner workings) in other microservices	(Conte, S.D., Dunsmore, H.E., Shen, 1986; Isoyama et al., 2012; Kluza et al., 2014; Kluza & Nalepa, 2012; Peltz, 2003)
Dependency	✓ More coupling	✓ No coupling or loosely coupling	(Conte, S.D., Dunsmore, H.E., Shen, 1986; Kluza et al., 2014; Nkomo & Coetzee, 2019)

State of the Art in Complexity Metrics

There have been many studies on measuring and evaluating the quality of software products using different metrics. Nkomo and Coetzee (2019) 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. 6), "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".

Similarities Between Metrics in Software Engineering and Business Process Modeling

Software engineering metrics have been used to evaluate different features of software products such as error prediction (Conte, S.D., Dunsmore, H.E., Shen, 1986), measuring the quality of software processes (Wang et al., 2011), measuring software functional size (Rolón et al., 2006), and quality metrics in software design (Monsalve et al., 2011). 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 software development project by both software engineers and business analysts to document and gather system requirements (Rolón et al., 2006). Hence, several studies have focused on finding similarities between software and business process. Table 2 compares the similarities between software and business process based on the modules, elements and compositional structure used in both domains (Monsalve et al., 2011).

Table 2: Similarities between software and business processes (Monsalve et al., 2011)

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

According to Table 2, there are similarities between software programs and business process models, regardless of the modeling language being used (e.g., BPEL, EPC or BPMN). A software program is divided into modules or functions, which perform by obtaining some inputs and providing some outputs. Similarly, business process models use activities. Hence, the order by which an activity is executed in a process model is predefined using operators such as sequence, splits and joins, which is similar to how the modules and functions interact in a software program (Monsalve et al., 2011).

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. The state-of-the-art on complexity metrics in business process modeling is summarized in Table 3.

Table 3: State of the art on complexity metrics in BPM

Metric	Reference
Lines of Code (LOC): Counts the number of lines of code in software programs. Cardoso et al. (2006), use LOC to adapt three size metrics for BPM (J Cardoso et al., 2006): 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	(Jorge Cardoso, 2005)
Control-Flow Complexity metrics (CFC): This metric is measured based on XOR-splits, OR-splits, and AND-splits in one process.	(J Cardoso et al., 2006; Jorge Cardoso, 2005)
McCabe's cyclomatic complexity metric (MCC): This 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.	(Banerjee, 2018; Jorge Cardoso, 2005)
Durfee square metric (DSM) and perfect square metric (PSM) • **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	(Kluza et al., 2014; Kluza & Nalepa, 2012)

times (each). • **PSM:** Is the (unique) largest number such that the top p types occur (together) at least $\lceil p^2 \rceil$ times, given a set of element types ranked in decreasing order of the number of their instances.	
I*nformation flow metrics by Henry and Kafura*: This approach focuses on evaluating the procedure complexity (PC) based on the frequency of calls in and out of the modules in one system	(Banerjee, 2018; Solichah et al., 2013; Vanderfeesten et al., 2007)
T*he 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.	(Banerjee, 2018; Kaimann, 1974; Mccabe, 1976; Sánchez-González et al., 2010)
C*onnectivity level between activities (CLA):* *This i*s measured by counting the total number of activities (TNA) divided by the total number of sequence flows between activities	(Banerjee, 2018; Wang et al., 2011)
H*alstead-based process complexity:* *This m*easures the complexity of business process models by using four measures (n1, n2, N1, and N2) to evaluate process length; process volume; and process difficulty. • 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)	(Banerjee, 2018; Latva-Koivisto, 2001; Solichah et al., 2013)
S*tructural metrics:* These metrics measure the level of understandability and error-probability in a model using multiple square metrics.	(Fitzsimmons and Love, 1978)

MICROSRVICE ORCHESTRATION and CHOREOGRAPHY MODELING

In this section we discuss the modeling of e-commerce scenarios using Zeebe Modeler BPMN 2.0 and Zeebe Simple Monitor (Zeebe.io, 2021). Each model depicts a single e-commerce scenario (i.e., workflow). We compose each workflow using both composition styles (i.e., Orchestration and Choreography), which rely on different communication mechanisms. For Orchestration, we use the Intermediate message catch event which relies on a send/receive message approach managed by the orchestrator to call each microservice and should wait for a response from the microservice to be able to call the next microservice. For Choreography, we implement event Choreography which uses microservice calls as events. This way microservices work independently without depending on calling other services.

Define Scenarios

We classify multiple scenarios based on different shopping processes on e-commerce websites. 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, namely small, mid-sized, and end-to-end; 3. Use BPMN 2.0 to model Choreography and Orchestration for each scenario from the 3 categories. We will use these categories later to evaluate the impact of the size of the models on their level of complexity.

Design BPMN Models

We use Zeebe BPMN Modeler, Zeebe-docker, CAMUNDA automation engine (Zeebe.io, 2021), and Amazon Web Services-AWS cloud to model the 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 (Zeebe.io, 2021) to model our workflows into microservice Choreographies and Orchestrations, representing the communication mechanisms among microservices and characteristics of each composition style.

Small-sized BPMN Workflows

In this section, we design BPMN models to capture the Choreography and Orchestration of microservices. Each model illustrates one single e-commerce module which uses one or group of microservices to perform a single task for an e-commerce application. These modules are user authentication, shipment, and payment.

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 1 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 login task to complete the process. In this scenario, four microservices are involved: browse site, login, sign up, and proceed as a guest.

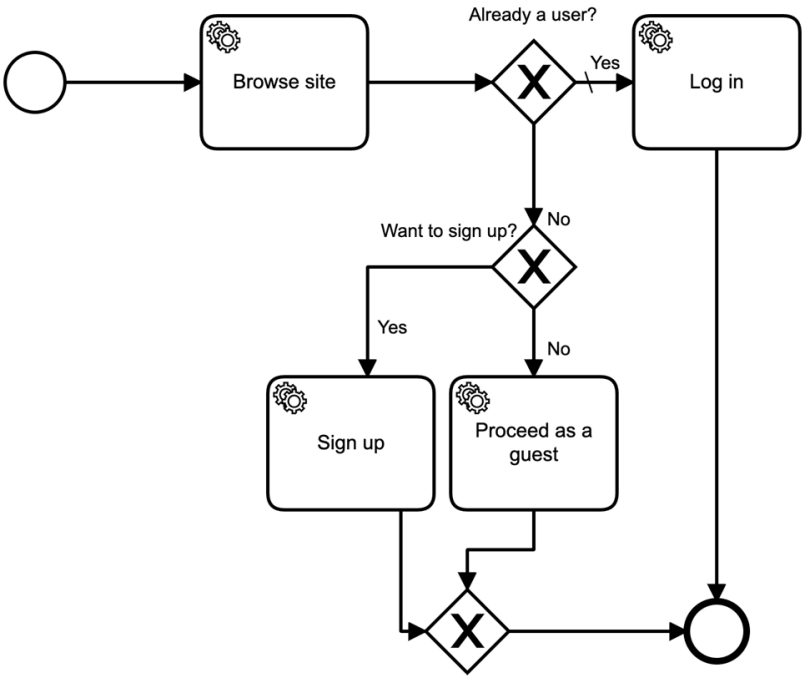

Figure 1: Choreography model for user authentication workflow

The second workflow, as shown in Figure 2, uses Orchestration to model the same scenario. To implement Orchestration for this workflow, we have defined a microservice task called e-commerce, which acts as the orchestrator. In order to deploy and execute the workflow on Zeebe Simple Monitor (Zeebe.io, 2021), we need to define an automated logic to call the next microservice 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 microservice in the process by giving it a predefined value. We use the name of each microservice 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. In this scenario there are six microservices involved: browse site, e-commerce (the Orchestrator), user authentication, login, sign up, and proceed as a guest.

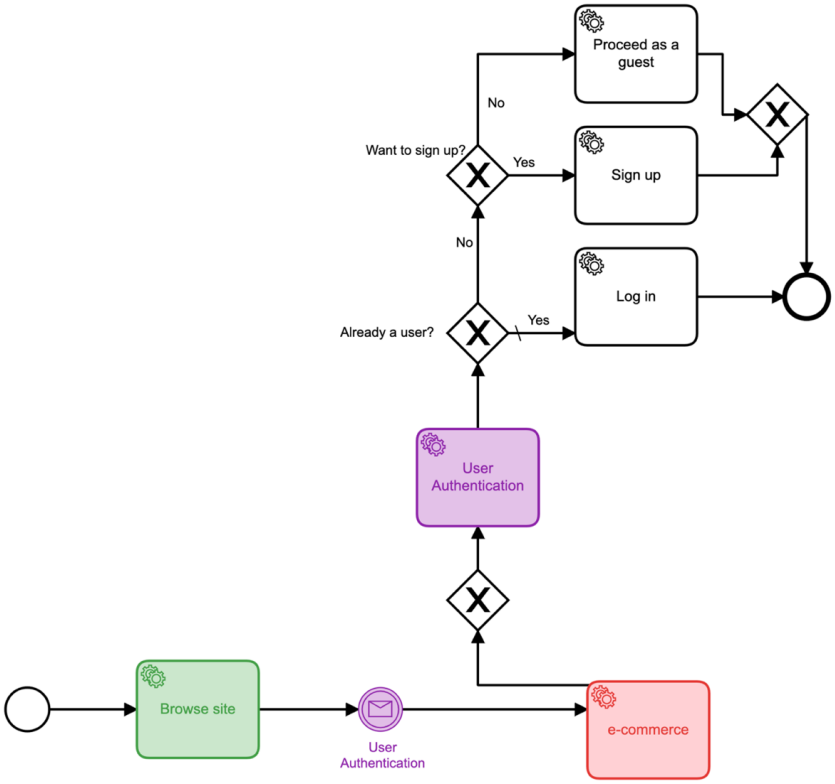

Figure 2: Orchestration model for user authentication workflow

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 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. To implement, 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 3 shows the shipment workflow as a Choreography. In this scenario there are six microservices involved: select delivery method, ship to my address, pick up at store, select the nearest store, add shipping fees, ship for free.

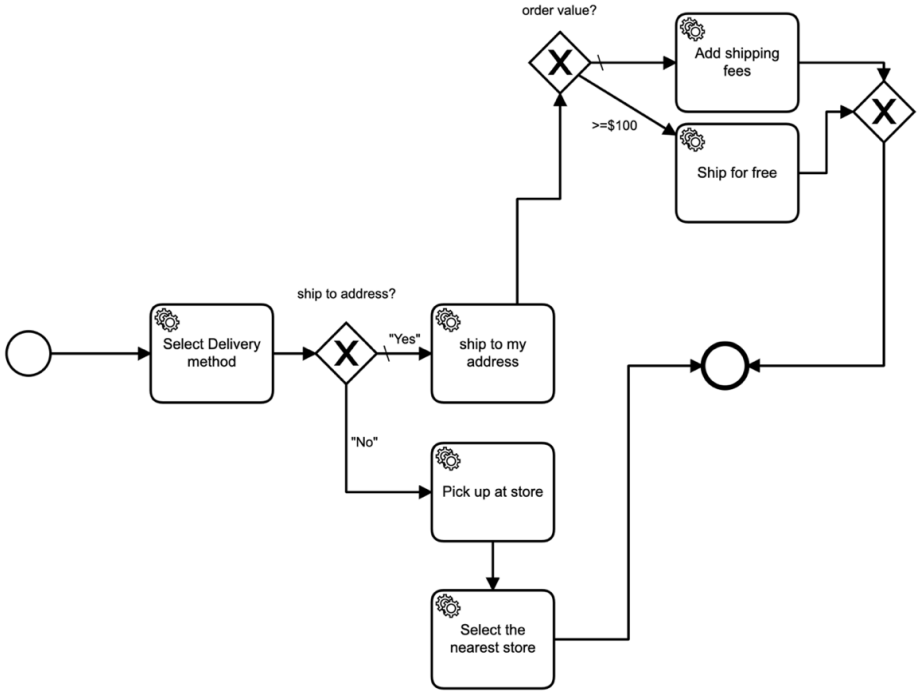

Figure 3: Choreography model for shipment workflow

The shipment workflow modeled as an Orchestration is shown in Figure 4. In this scenario there are eight microservices involved: e-commerce (Orchestrator), shipment, select delivery method, ship to my address, pick up at store, select the nearest store, add shipping fees, ship for free.

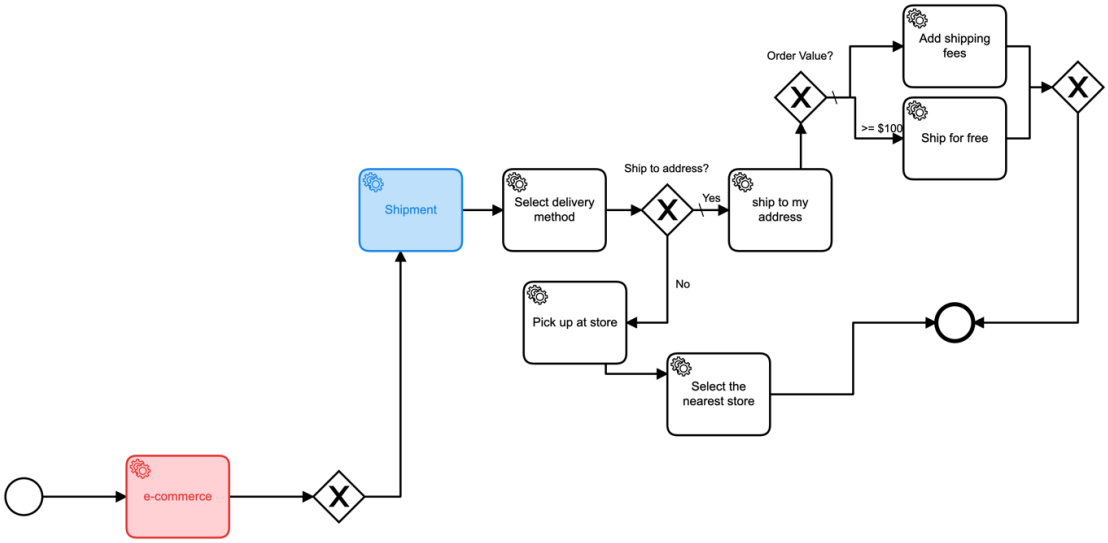

Figure 4: Orchestration model for shipment workflow

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 5 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. In this scenario there are two microservices involved: initiate payment and choose payment method.

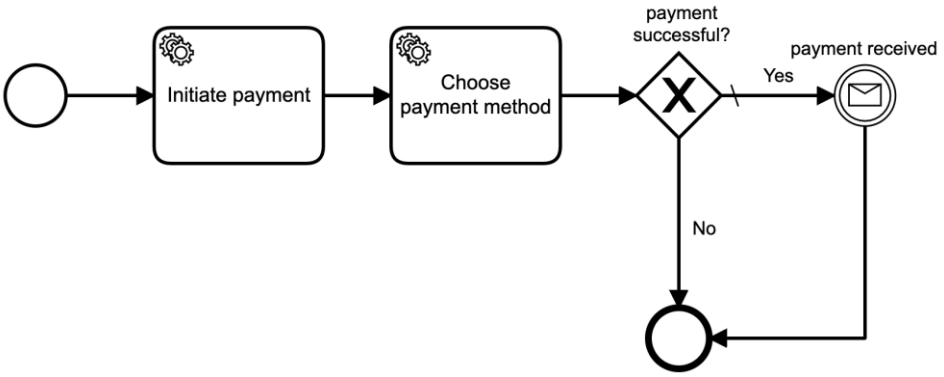

Figure 5: Choreography model for payment workflow

The payment workflow modeled as an Orchestration is shown in Figure 6. As mentioned above, e-commerce is the central controller in this process. In this scenario there are three microservices involved: e-commerce (Orchestrator), initiate payment, choose payment method.

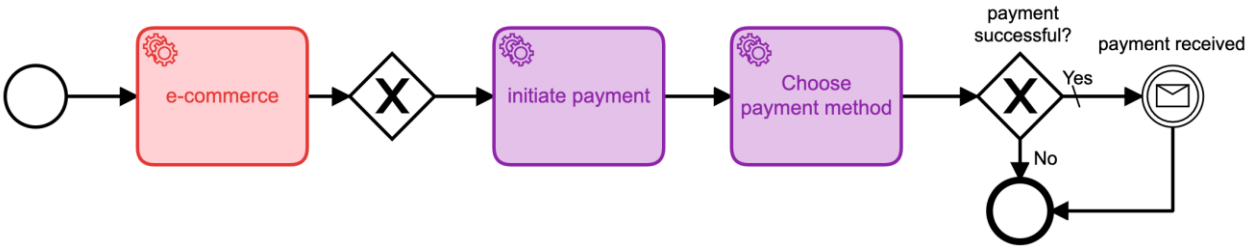

Figure 6: Orchestration model for payment workflow

Mid-sized Workflow

In this section, we combine the workflows described above to design a new set of models, which include more than one microservice in their process to perform the user authentication and shipment modules of an e-commerce application as part of the checkout process, Figure 7 and Figure 8. We use the same tools for the design, deployment, and execution of the models (Zeebe.io, 2021).

User Authentication + Shipment

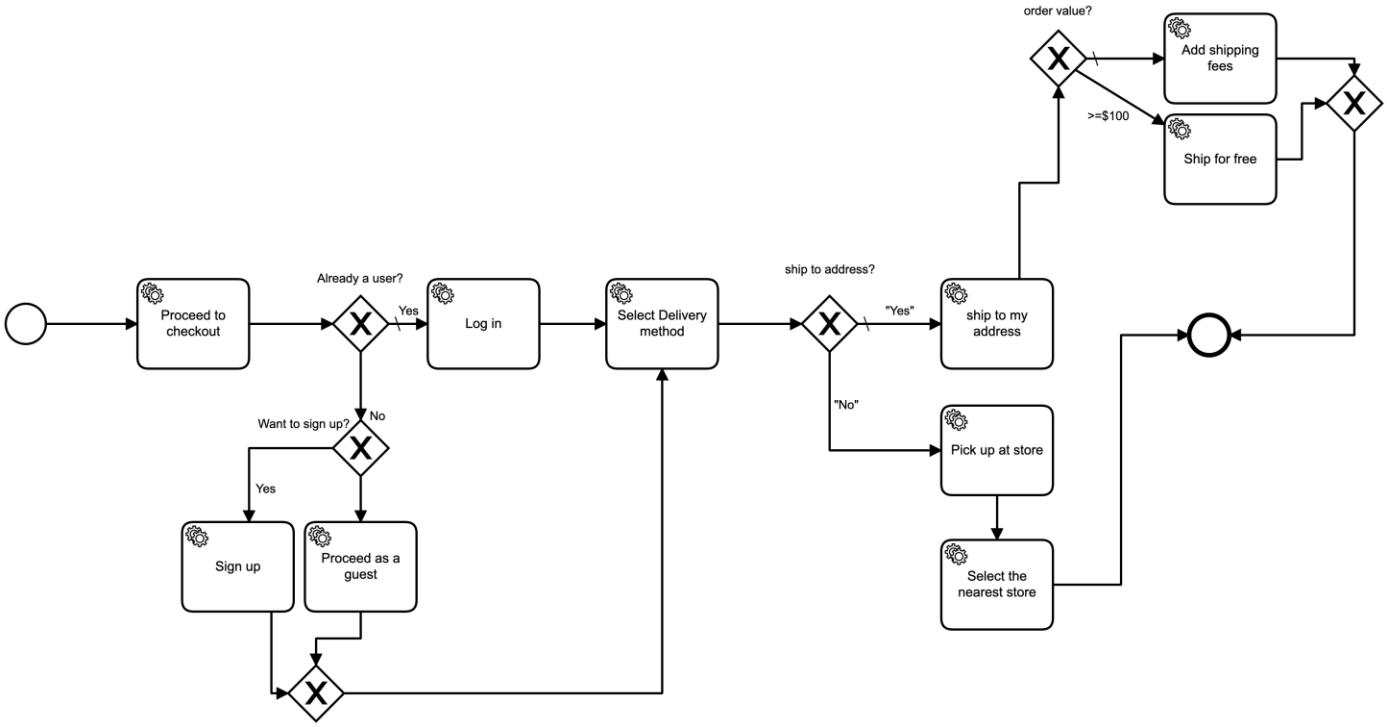

Figure 7: Choreography model for user authentication +shipment workflow

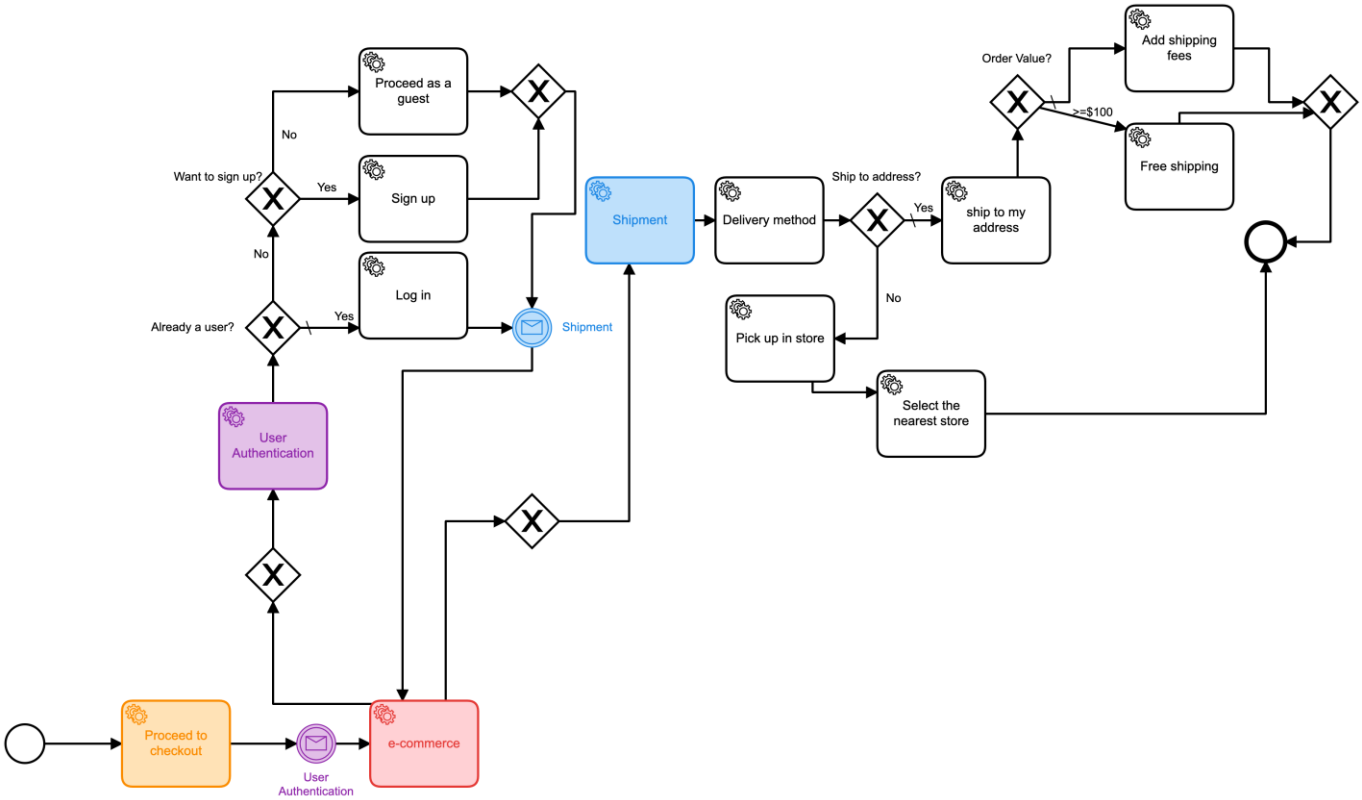

Figure 8: Orchestration model for user authentication + shipment workflow

End-To-End Workflow

We designed an end-to-end workflow for an e-commerce application from when a user starts browsing on the website until the order is delivered. We use the same modeling technique and tools to design the Choreography and Orchestration in Figure 9 and Figure 10 to illustrate a full lifecycle of an e-commerce application.

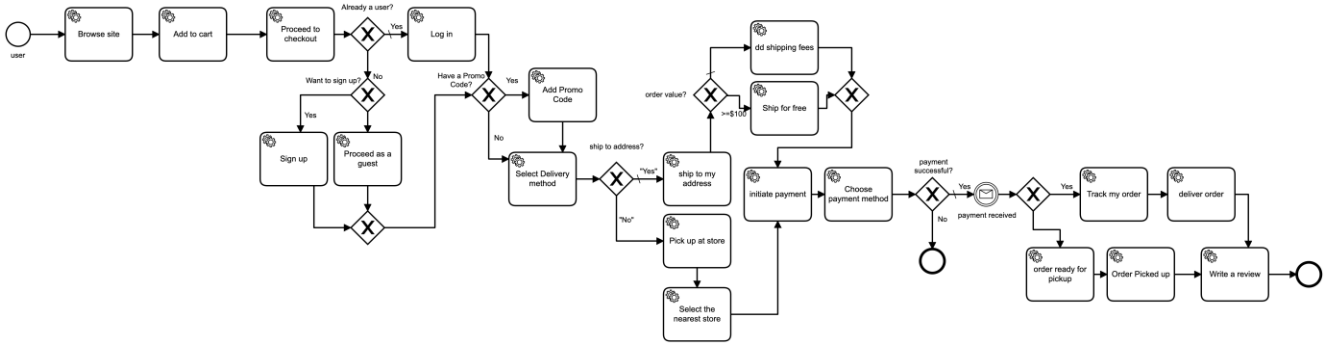

Figure 9: Choreography model for end-to-end workflow

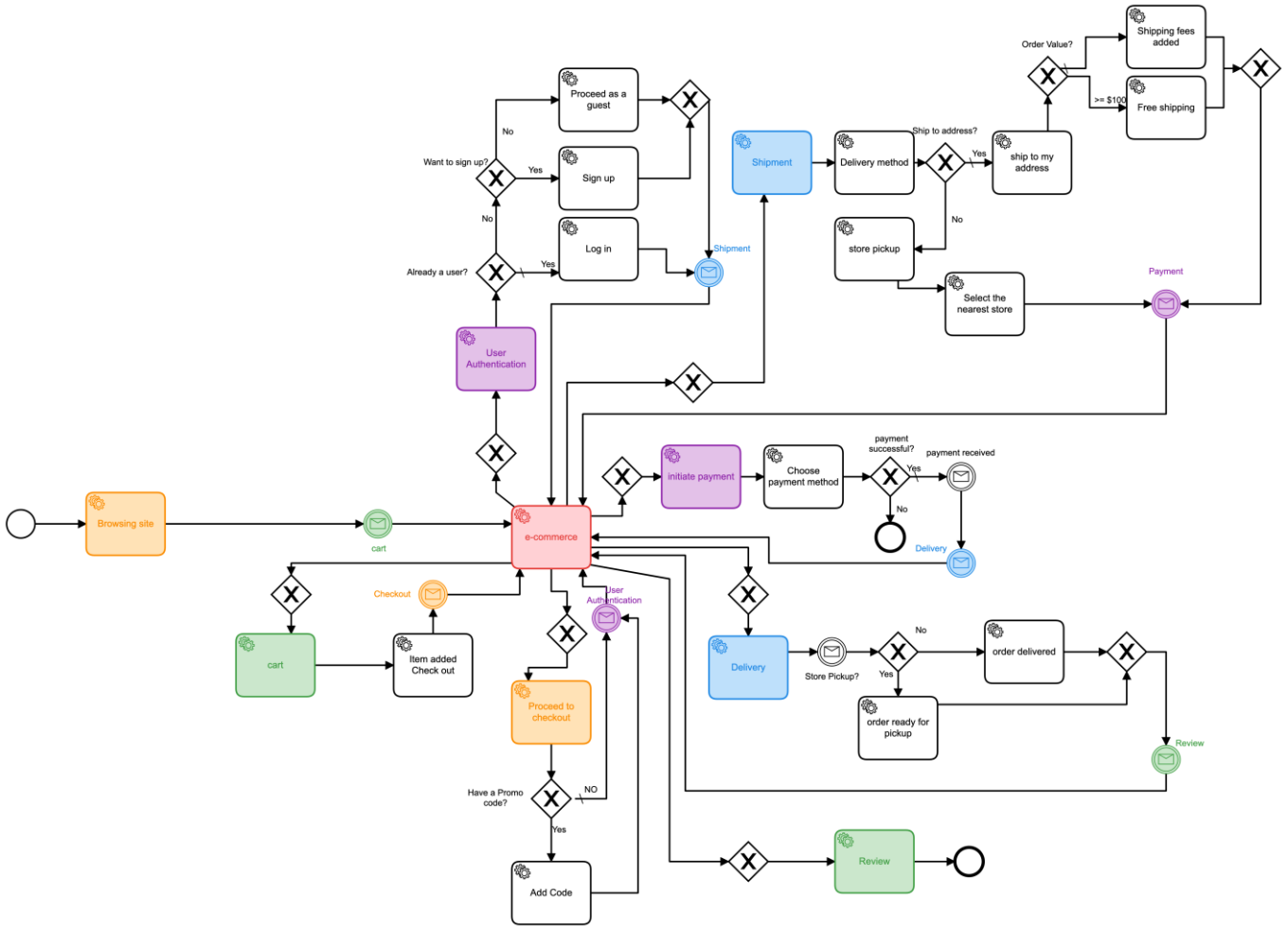

Figure 10: Orchestration model for end-to-end workflow

IMPLEMENTATION & EVALUATION

In this section, we first present the approach and tools we use to deploy the resulting models for both Choreography and Orchestration and create and execute instances of each model following the guidelines provided in (Zeebe.io, 2021). Next, we use the metrics from Section 3 to assess the complexity of each workflow.

Implementation Tools

We use Zeebe Modeler to design our BPMN models, and Camunda Workflow to deploy those models on a platform called Zeebe Simple Monitor. This platform gives access to a consistent environment to deploy and run BPMN models, using AWS cloud as the server, which we use to upload and execute our models to test if there are any errors with the workflows (Zeebe.io, 2021). 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.

Evaluation of Complexity

In this section we describe how we apply the complexity metrics discussed in Section 3 to measure the complexity of our models. BPMN is used to capture business process flows using flowchart-based steps. BPMN consists of four groups of elements for modeling, which are Flow objects, Connecting objects, Swimlanes, and Artifacts.

- Flow Objects: Include activities, events, and gateways
- Connecting Objects: Include sequence flow, message flow, and association
- Swimlanes: Include lanes and pools to separate activities into different units
- Artifacts: Include data objects, group, and annotation

All the above-mentioned elements can be used when modeling BPMN workflows, however, based on the modeling logic that is being implemented, some of the elements may not serve the purpose of the designed models. This modeling logic can be affected by various factors, including the domain of the business, the modeling tool, and the technology, which might impact the modeling patterns and results considering what measurement technique is being used.

In our research, we focus on e-commerce as our domain, using Zeebe Modeler as the main modeling tool and microservices as the technology to design BPMN models. Hence, given the logic we use for our designed workflows, not all the metrics are applicable to be used on 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, since we are not focusing on the data model design and data sharing architecture of microservices. Hence, this metric cannot be used for our evaluation.
In the following, we use the shipment workflow (see Figure 3 and Figure 4) as our sample model to explain how we apply the complexity metrics discussed in the previous chapter and compare the level of complexity between them. After that, we apply the metrics to the remaining models to measure their complexity.

Lines of Code Metric
The lines of code metric can be used to measure the complexity of a process. Zeebe modeler generates XML code for each BPMN model. For the Shipment workflow, 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.

Size Metrics
We apply the three size metrics proposed in (Vanderfeesten et al., 2007) from Section 2.5 to the Shipment workflow and the results are shown in Table 4.

Table 4: The size metrics for Choreography and Orchestration

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

Control-Flow Complexity Metrics (CFC)
We apply the CFC metric to the Choreography and Orchestration of the Shipment workflow. 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 3, in the Choreography workflow, there are 5 outgoing arcs in total, so the CFC equals 5. And according to Figure 4, for Orchestration the CFC equals 6, which shows a higher level of complexity in Orchestration.

Durfee Square Metric (DSM) and Perfect Square Metric (PSM)
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 shipment workflow with the number of times they occurred to find the element with the least frequency. As shown in Table 5, the DSM for Choreography is lower than Orchestration which depicts that there is more complexity in Orchestration as the result of this measurement.

Table 5: 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 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.

Coefficient of Network Complexity Metrics
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 Figure 3 and Figure 4 and the results are shown below in Table 5. 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 6, Orchestration has a higher level of complexity compared to Choreography.

Table 6: The Coefficient of Network Complexity metric

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

Structural Metrics

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 2. As shown in Figure 3, the Diameter for Choreography is 7 while the Diameter measured for Orchestration (Figure 4) 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 (Fitzsimmons and Love, 1978).

RESULTS

We applied all the above-mentioned metrics on all our designed workflows, and we summarised the results in Table 7 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. For instance, when we observe the results from the CFC metric, we notice that the level of complexity of Orchestration is greater than the Choreography and as the size of the model gets bigger the level of complexity increases so we see more difference in the numbers of the level of complexity of the end-to-end Orchestration model compared to the End-to-end Choreography model. The same is true for the number of activities involved in each composition technique, so from the modeling experience we also realized that there are more activities involved in Orchestration type workflows which makes the modeling process longer and more complex.

The results from the measurement of the Diameter also prove a higher level of complexity in Orchestration 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. So overall, our results show a higher level of complexity in Orchestration than Choreography.

Table 7: 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
Number of activities	User Authentication	4	6
	Shipment	6	8
	payment	2	3
	User Authentication +Shipment	10	13
	Shipment+ Payment	11	12
	End-to-end	20	23
	User Authentication	6	10
	Shipment	7	10

Diameter	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

CONCLUSION & FUTURE WORK

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 uncovered the differences between microservice Orchestration and Choreography. Secondly, 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 question announced in Section 1 as follows:

- Which composition technique (Orchestration or Choreography) is less complex to deliver business requirements in e-commerce applications based on the proposed scenarios? The 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 discussed how complexity can affect the modifiability and understandability of each composition style, which makes Choreography models more modifiable and understandable compared to Orchestration.

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.

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, as suggested by Zeebe.io (Zeebe.io, 2021). The tool enables us to test the applicability of our models to 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 between microservices via using conditional sequence flows and XOR gateways offered by Zeebe Modeler BPMN 2.0 in our workflows.

Limitations

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 may vary depending on what modeling technique and tool is used. The second limitation of this study is that there are some complexity metrics that 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 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.

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

Banerjee, S. (2018). Concepts and Tools for Measuring the Complexity of Service Choreography Models [University of Stuttgart]. https://elib.uni-stuttgart.de/bitstream/11682/10041/1/SayanBanerjeeMasterThesis.pdf

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

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.

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.

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

Conte, S.D., Dunsmore, H.E., Shen, V. . (1986). Software engineering metrics and models. Benjamin-Cummings Publishing Co., Inc.

Fitzsimmons, A., & Love, T. (1978). A review and evaluation of software science. *ACM Computing Surveys (CSUR)*, *10*(1), 3-18. https://dl.acm.org/doi/10.1145/356715.356717

Haouari, G., & Ghannouchi, S. A. (2017). Quality Assessment of an Emergency Care Process Model based on Static and Dynamic Metrics. Procedia Computer Science, 121, 843–851. https://doi.org/10.1016/j.procs.2017.11.109

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

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, R. A. (1974). Coefficient of network complexity. *Management Science*, *21*(2), 172-177. Stable URL : https://www.jstor.org/stable/2629677

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

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

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

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 (first ed.). O'Reilly Media, Inc. 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

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., Cardoso, J., García, F., Ruiz, F., & Piattini, M. (2009). Analysis and validation of control-flow complexity measures with BPMN process models. Lecture Notes in Business Information Processing, 29 LNBIP(4), 58–70. https://doi.org/10.1007/978-3-642-01862-6_6

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. (2018). 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

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

Shrestha, H., & Vuorimaa, P. (2019). Design Science Methodology for Microservice Architecture and System Research [Aalto University]. https://aaltodoc.aalto.fi/bitstream/handle/123456789/39157/master_Shrestha_Hari_2019.pdf?sequence=1&isAllowed=y

Singhal, N., Sakthivel, U., & Raj, P. (2019). 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

Valderas, P., Torres, V., & Pelochano, V. (2020). Supporting a hybrid composition of microservices. The eucaliptool platform. *Journal of Software Engineering Research and Development*, *8*, 1-1. https://doi.org/10.5753/jserd.2020.457

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

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

Zeebe.io. (2021). Introduction - Zeebe Documentation. https://stage.docs.zeebe.io