2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) | 978-1-6654-2705-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/SEAA53835.2021.00038

Event Oriented vs Object Oriented Analysis for Microservice Architecture: An Exploratory Case Study

Huseyin Unlu
Computer Engineering Department
Izmir Institute of Technology
Izmir, Turkey
huseyinunlu@iyte.edu.tr

Samet Tenekeci
Computer Engineering Department
Izmir Institute of Technology
Izmir, Turkey
samettenekeci@iyte.edu.tr

Ali Yıldız
Computer Engineering Department
Izmir Institute of Technology
Bilgi Grubu
Izmir, Turkey
ali.yildiz@bg.com.tr

Onur Demirors
Computer Engineering Department
Izmir Institute of Technology
Izmir, Turkey
onurdemirors@iyte.edu.tr

***Abstract*—The rapidly developing internet infrastructure together with the advances in software technology has enabled the development of cloud-based modern web applications that are much more responsive, flexible, and reliable compared to traditional monolithic applications. Such modern applications require new software design paradigms and architectures. Microservice-based architecture (MSbA), which aims to create small, isolated, loosely-coupled applications that work in cohesion, becoming widespread as one of these approaches. MSbA allows the developed applications to be deployed and maintained separately, as well as scaled on demand. However, there is no de facto method for the analysis and design of systems for these architectures. In this paper, we compared the usefulness of the object-oriented (OO) and event-oriented (EO) approaches for analyzing and designing MS-based systems. More specifically, we performed an exploratory case study to analyze, design, and implement a software application dealing with the 'application and evaluation process of graduate students at IzTech'. This paper discusses the results of this case study. We observe that the EO approaches have significant advantages with respect to the OO approaches.**

Keywords— microservices, event-driven process chains, eEPC, event-oriented analysis, cloud

I. INTRODUCTION

The growing capabilities of the cloud, the need for continuous digital transformation together with the recent developments in software architectures, and enabling technologies enabled the development of modern applications that are much more responsive, flexible, and reliable. New software design paradigms and architectures that focus on creating cloud-compatible applications are required to meet these new demands. MSbA (Microservice-based Architecture) is becoming widespread in leading software design companies as one of these approaches [1].

An MS-based system consists of multiple microservices that are highly cohesive and loosely coupled [2]. The degree of functional dependence among the elements within each microservice is high, while the degree of interdependence among microservices is as low as possible. In such a system, each microservice is designed as an isolated, autonomous application with a small bounded context and a single responsibility [3]. On the other hand, they can cooperate and coordinate with other microservices (i.e. they are composable) through asynchronous communication channels in order to perform more complicated tasks. This granular structure provided by MSbA enables the applications to be deployed, scaled, and tested independently, and thus it increases the scalability and fault-tolerance of the system.

Designing a single microservice might be relatively straightforward but does not mean a lot. On the other hand, the design of MS-based systems is a great challenge for software companies as there is a lack of well-defined methods proposed for the analysis and design of MSbAs. Analysis and design for MS-based systems are frequently being performed by ad-hoc methods [4] and if a method is used, it is usually based on Object-Oriented Analysis and Design (OOAD). However, the structural decompositions required by microservices are quite different from those of OOAD methods. Alternative approaches such as Event Storming [5], [6] lack the elements of a typical method, such as systematic implementation process, modeling notations to be used in different phases, and software tools to support the process.

In this study, we explore a novel approach for MS-based analysis and design (MSbAD). We applied Event-Oriented (EO) and Object-Oriented (OO) methods to analyze a problem to develop an MSbA solution. In accord with the methodologies used, we used Extended Event-Driven Process Chains (eEPC) for EO methods and traditional UML notations including use case, class, sequence, and activity diagrams, for comparison. The exploratory case study included analysis, design, and implementation of the system for the 'application and evaluation process of graduate students at IzTech' *(https://lee.iyte.edu.tr/en/graduate-application/application-to-masters-programs/).* First, we performed Event-Oriented Analysis and Design (EOAD) and implemented an MSb solution for the case. Secondly, we performed OOAD to decompose the problem into microservices. To prevent possible bias, an independent group also performed OOAD for MSb decomposition. We then compared the advantages and difficulties of each approach with respect to the commonsense MSbA principles.

II. Related Work

The concept of service represents an independent business function with a clear purpose. Services should provide functionality and be enabled through a common interface protocol. The behavior of the service is clearly defined to its user, but the development details are hidden. Thus, only the purpose and result of the service call are visible [7].

Service-Oriented Architecture (SOA) is an architectural approach based on the principle of separation of concerns that has been widely used in software projects since the early 2000s. The evolution of cloud architectures, serverless systems, applications based on large data processing, and the demands that transform scalability and availability to the most basic requirements, stipulated the rapid change of classical SOA architectures [8].

MSbA is a new approach to create distributed software systems. It focuses on the design and development of maintainable, easily scalable, and highly available systems [9]. Services in MSbA are loosely coupled independent parts that communicate with each other over the network to achieve a goal. These microservices can communicate with technology-independent protocols so that the choice of development languages or platforms becomes the only issue of the service. They can be changed independently and interruptions in their operation do not cause interruption of the entire system.

Although microservice is frequently thought of as a small service, it has fundamental differences. The most obvious difference is to eliminate coupling on objects, stop using the database for coordination and enable communication between services asynchronously. Microservices use lightweight protocols for communication such as HTTP, REST. Each microservice should be changeable without impacting the operation and performance of the other and independently deployable. Application servers are usually not used. It is common to use cloud platforms. These basic needs require changing the analysis and design approach of software and data handling.

MSbAD approaches in the literature can be classified into two; transition to microservices from a monolithic application [10]–[12] and developing a new set of applications as a system of microservices [13]–[15].

Li et al. [10] proposed a dataflow-driven semi-automatic decomposition approach to identifying microservices from monolithic applications. In this approach four-step decomposition procedure is introduced and the method heavily relies on detailed DFDs at different levels. The quality of DFDs is critical for the decomposition results.

Kamimura et al. [11] proposed a method that identifies the candidates of microservices from the source code by using the software clustering algorithm SArF with the relation of 'program groups and 'data' which it is defined.

Al-Debagy et al. [12] proposed a new decomposition method for designing microservices. They modify monolithic application to a microservice application by analyzing the application programming interface by using word embedding models to obtain word representations using operation names, as well as, using a hierarchical clustering algorithm to group similar operation names together in order to get suitable microservices.

Ma et al. [13] proposed a graph-based and scenario-driven approach to the development of MS-based systems, referred to as GSMART (Graph-based and Scenario-driven Microservice Analysis, Retrieval, and Testing). It enables the automatic generation of a 'Service Dependency Graph (SDG)'. Users are expected to follow defined guidelines to allow GSMART to retrieve required information for producing and analyzing SDGs.

Santos et al. [14] proposed a model-based approach for designing a logical view of a microservice architecture (MSA), called 4SRS-MSLA. The approach is based on modeling a business domain in UML use cases, thus deriving a UML component diagram for the domain, and finally grouping the components into microservices.

Baresi et al. [15] used interface analysis for microservice identification. Their solution is based on the semantic similarity of foreseen or available functionality described through OpenAPI specifications. This approach relies on well-defined and described interfaces that provide meaningful names, and follow programming naming conventions.

The Event Storming Method was introduced by Brandolini [5]. It is a workshop format in which participants from different areas work together. It enables the exploration of business domains by focusing on domain events that are essential for MSbA, generated in the context of a business process. The Event Storming workshop requires physical space with sticky notes, a pen, and huge whiteboards. While local teams can work well in this physical space, it can be difficult for remote teams. Besides, documenting all knowledge put on the whiteboard can be difficult. The involvement of all domain experts is desired but more participation becomes coordination more difficult.

Considering the available literature, the suggested approaches are usually based on OOAD and identify microservices using a clustering algorithm, graph-based analysis, and interface analysis from a monolithic application. Microservices have distinct properties such as the bounded context in relation to events, asynchronous communication, and event-based logging. OOAD produces class-based decomposition and it is difficult to find events and define bounded context based on events. Identifying microservices from monolithic application-based methods requires completely different viewpoints that are not produced during the design representation.

III. Research Methodology

The analysis of a problem for designing a solution to an MSbA is a challenging process. In this study, our goal is to observe the needs of microservice analysis and design approaches. The literature review showed that there is no de facto method for MSbAD and most of the ad-hoc approaches lack the elements of a methodology. Thus, we aim to answer if the OO or EO analysis approaches provide a good strategy for developing an MS-based solution.

An MS-based solution is required to have 10 important characteristics: loose coupling, cohesion, isolation, autonomy, composability, small bounded context, single responsibility, scalability, fault tolerance, and asynchronous communication [2], [3], [9], [16]. We evaluate two approaches in this context. Thus, our research question is as follows: "How successful are OOAD and EOAD in meeting the important characteristics required by an MSbA?". To answer our research question, we apply our research methodology which includes 5 phases: the selection of the case, performing EOAD, performing OOAD, performing OOAD by an independent group, and comparison of the approaches for our case (Fig. 1).

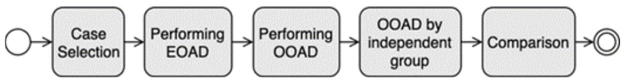

Fig. 1. Research Methodology

The selected case must meet a number of criteria. Firstly, it must include processes that can be automated. Secondly, there must be specific events that will trigger each process. Thirdly, the case should be large enough to implement at least two microservices but should be small enough to be implemented as a whole in a reasonable time. Finally, we should be familiar with all the steps of the process so that we can work on the case without struggling with problem domain details.

IV. Case Study

As the first phase, we selected "the graduate student application and evaluation process at IzTech" as a case that satisfies all four criteria mentioned in the previous section: (1) the process is suitable for distributed execution and automation since it consists of relatively small and isolated subprocesses such as application, verification, evaluation, and notification, (2) all subprocesses can be triggered through predefined events and executed in cohesion, (3) the scope can be narrowed or enlarged depending on time and resource limitations, and (4) the whole process can be easily modeled by using the explicit instructions on university's web page *(https://lee.iyte.edu.tr/en/graduate-application/application-to-masters-programs/).*

A. Event-Oriented Analysis and Design (EOAD)

The second phase of our research methodology is to perform EOAD for the case. The methodology called EOAD is defined as part of a graduate course in IZTECH: "CENG 555 Analysis and Design of Microservice Based Systems"" (https://ceng.iyte.edu.tr/courses/ceng-555/)'. The methodology includes 4 iterative steps (Fig. 2). First, an EO problem analysis is performed. The current business processes (AS-IS) are modeled using eEPC notation. eEPC mainly consists of events, functions, and connectors [17]. Each process starts and ends with an event. In the flow, each function is triggered by an event. After the high-level process is modeled, the detailed subprocesses are presented in low-level eEPC diagrams. eEPC diagram of the AS-IS process for our case is presented in Fig. 3. The process starts with an application announcement and is followed by the application of the student, verification of the application, interview with the applicant, assessment of the interview, sending the assessment result to the graduate school, and announcement of the results on graduate school's web page, sequentially. The low-level subprocesses are hidden in the AS-IS diagram to increase clarity. Processes with detailed eEPC diagrams are indicated with a '(+)' sign and listed at https://bit.ly/3iplq6u.

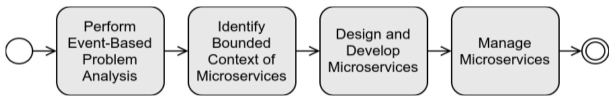
Perform Event-Based Problem Analysis
Identify Bounded Context of Microservices
Design and Develop Microservices
Manage Microservices

Fig. 2. EOAD Methodology

The second step is to identify the bounded context of the microservices. This step is performed by drawing the TO-BE model where the process becomes automated. The responsibilities of the software are specified. Similar to the AS-IS model, the details can be presented at different levels to provide simplicity in the diagram. In this step, events are shown in eEPC where they are located inside boundaries. TO-BE model for our case is presented in Fig. 4. The graduate school's announcement of application acceptance and the application of the prospective applicants are performed as same as the AS-IS model. Differently, the initial application process is managed by the application microservice and the applicant is informed about the status of his/her application through notification microservice. Additionally, the application requirements and documents are verified automatically by verification microservice as well as manually by the graduate school. After the two-step verification, the interview and exam lists are generated. The department is responsible for holding and assessment of the examinations and entering the results into the system. The entered assessment results need to be confirmed by the graduate school. The assessment and confirmation processes are handled by an evaluation microservice. The last step is the announcement of the application results via the system. When it is performed by the graduate school, the applicants are automatically informed through the notification microservice. Similar to the AS-IS model, the low-level subprocesses are hidden in the TO-BE diagram to increase clarity. Processes with detailed eEPC diagrams are indicated with a '(+)' sign and listed at https://bit.ly/3iplq6u.

The third step is to design and develop the microservices considering design patterns. We designed four microservices to implement our case study based on our EO TO-BE model: application, verification, evaluation, and notification (Fig. 4). The context of these microservices is explained in the following section. The last step of EOAD is to manage the microservices. The operation of microservices is different from traditional development approaches. In this step, different methods such as DevOps should be evaluated. In this study, we do not focus on this step.

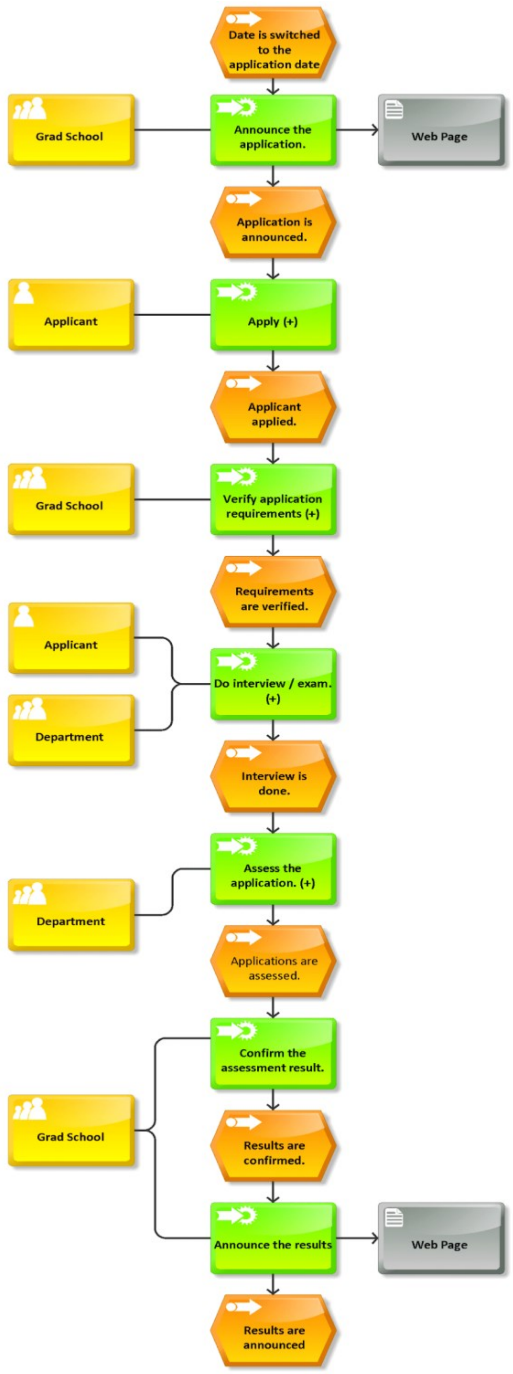
Date is switched to the application date
Grad School
Announce the application.
Web Page
Application is announced.
Applicant
Apply (+)
Applicant applied.
Grad School
Verify application requirements (+)
Requirements are verified.
Applicant
Do interview / exam. (+)
Department
Interview is done.
Assess the application. (+)
Department
Applications are assessed.
Confirm the assessment result.
Grad School
Results are confirmed.
Announce the results
Web Page
Results are announced

Fig. 3. eEPC diagram for the AS-IS model

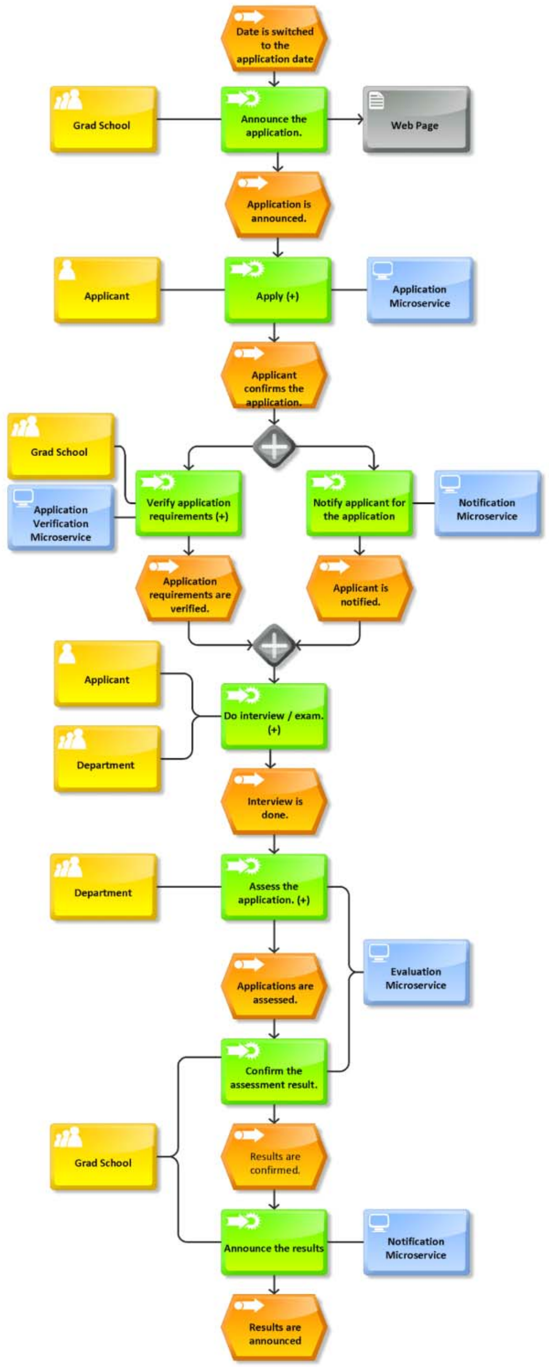
Date is switched to the application date
Grad School
Announce the application.
Web Page
Application is announced.
Applicant
Apply (+)
Application Microservice
Applicant confirms the application.
Grad School
Application Verification Microservice
Verify application requirements (+)
Notify applicant for the application
Notification Microservice
Application requirements are verified.
Applicant is notified.
Applicant
Do interview / exam. (+)
Department
Interview is done.
Department
Assess the application. (+)
Evaluation Microservice
Applications are assessed.
Confirm the assessment result.
Grad School
Results are confirmed.
Announce the results
Notification Microservice
Results are announced

Fig. 4. eEPC diagram for the TO-BE model

In EOAD, bounded context is identified by the events. An event triggers the microservice and upon the completion of it generates another event. The event draws the boundaries of the context. As the automated process is based on events and each bounded context generates an event and is triggered by an event, asynchronous communication is implemented. The details of the communication based on message queues are described in the following section.

System Architecture. We developed four microservices for the system using the eEPC diagram of the TO-BE model. A general overview of our MSbA is presented in Fig. 5. The context of each microservice is defined as follow:

- *Application Microservice:* The applicant applies to the graduate programs by this microservice. The related information is entered through a web user interface. After the applicant confirms the application, four operations are performed respectively: (1) application microservice creates a data object that consists of all application data and other fields (such as verification status, assessment status, and confirmation status) which will be filled by verification and evaluation microservices, later on, (2) the data object is converted to a JSON object and saved in the database, (3) a new message is generated, its headers and properties are set, and the JSON object obtained in the previous step is attached to its payload, (4) the message including the type and payload is added to the queue.

- *Verification Microservice*: The verification MS is subscribed to the message queue through a JMS Listener. Thus, it continuously listens to the messages sent by the application microservice. Each message enqueued by application microservice is dequeued by verification microservice. When a message is fetched and parsed, four operations are performed: (1) the JSON payload of the message is mapped to a data object, (2) the verification is performed on the data object and the verification status is set accordingly, (3) the updated data object is converted to a JSON object and saved in the database, (4) a new message containing the JSON object in its payload is created and added to the queue.
- *Evaluation Microservice:* Evaluation microservice is used by two actors: (1) the relevant department members who perform assessments for the applications, (2) the graduate school officers who confirm the assessments. Each message enqueued by verification microservice is dequeued by evaluation microservice. It is automatically parsed and stored in the database. The relevant department member or graduate school officer accesses the evaluation microservice through the web user interface. Thus, he/she views and updates the applications stored in the database. As soon as the assessment status or confirmation status of a selected application is updated four operations are performed respectively: (1) a new data object including the updated information is created, (2) the new data object is converted to a JSON object, (3) the corresponding JSON object in the database is updated, and (4) a new message containing the new JSON object in its payload is created and added to the queue.
- *Notification Microservice:* The notification microservice continuously listens to the messaging queue for any type of messages published by application, verification, and evaluation microservices. It makes an API call to the external mailing service to send notifications to applicants about the status of their applications. The notification content is structured according to the message type, which was previously set by the trigger microservice (application, verification, or evaluation).

Implementation Details. While implementing an MSbA, we have to choose either a multi-queue or a single-queue model to enable communication between the microservices. The multi-queue model is based on splitting a monolithic queue into single-purpose queues that pass a specific type of message between microservices. In such an architecture, no tagging or other additional control structure is needed while receiving messages, since any two microservices communicate through their own queue. The multi-queue model has its own advantages such as providing less complicated queues and simple message payloads. However, it has significant drawbacks such as potentially alternated data models, hardness in flow-control, and forming the queue explosion anti-pattern which causes multiplying the load and increasing the latency on message brokers. Thus, we have used a single-queue model that passes different types of messages through a singular queue. In such an architecture, all related microservices are subscribed to a generic queue but listens only for the specific events tagged for themselves. Here, the tag is a simple header field such as JMSType and is set by the microservice that publishes the message. The message payload contains a generic, unified data object which can be manipulated by different microservices. The single-queue-based architecture provides many advantages including unified object modeling, easy flow-control, decreased load, and latency on message brokers. However, it has some disadvantages such as overloaded message payloads and potential single point of failures arising out of the monolithic queue.

The microservices, discovery server, API gateway, and load balancer have been implemented using Spring Cloud Tools, which are part of the Spring Boot Framework (*https://www.spring.io/projects/spring-boot*). Spring Cloud includes Eureka Discovery Server for service registration and discovery, API gateway for dynamic routing of direct API calls from users or other services, the Load Balancer for automatic load balancing, and many other out-of-box tools for developers to build applications running on the cloud. The remaining part of the system consists of a web user interface, Google Mail API (*https://developers.google.com/gmail/api/guides/*) as an external mailing service, mongoDB database (*https://www.mongodb.com*) to store JSON-formatted data for each microservice, and Apache ActiveMQ Message Broker (*https://activemq.apache.org/*) as the message queue to establish asynchronous communication between microservices through a message queue. All of the source codes are available at https://github.com/smtnkc/ceng555.

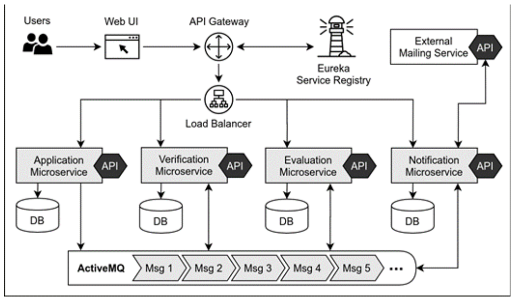

Fig. 5. System overview for the microservice-based architecture

B. *Object-Oriented Analysis and Design (OOAD)*

The third phase is to perform OOAD for the selected case. This phase aims to see how OOAD can be applied in MSbAs. We performed OOAD for the selected case. However, the activities of the second phase (EOAD) could have given some insights that changed how the third phase (OOAD) was performed. Thus, it can create an experimental bias. To prevent potential bias, OOAD for the selected case was also performed by an independent group as the fourth phase. We performed OOAD at a high level. On the other hand, the independent group performed OOAD in finer granularity. This section includes the analysis and design of both groups.

OOAD performed by authors. First, we identified the use cases (Fig. 6). Although the use case diagram is successful in defining most of the main user activities, it is insufficient in showing external-actor and system events such as notification. Then, we identified the classes (Fig. 7). The class diagram provides a detailed representation of the main elements and the relations between them. Although this is very useful to build a well-organized data model, it does not provide any information about system behaviors. On the other hand, the system behaviors can be tracked on the activity diagram (Fig. 8). Although the activity diagram explicitly visualizes the general workflow, it does not cover the external actors together with their respective functions as well as the resources and information used by each function. In summary, OOAD could not provide us a natural decomposition strategy for developing an MS-based solution. We could not identify the microservices for the case based on the diagrams. Indeed, OOAD provided a class-based decomposition strategy which does not help to identify the boundaries of the microservices. We have to identify microservices by using a different viewpoint not produced in OOAD naturally.

OOAD performed by the independent group. We asked the third-year students who are taking a Software Engineering Course to analyze and design the selected case applying a traditional object-oriented approach and then implement a solution for the case. For this purpose, they prepared two different documents based on IEEE standards: IEEE Software Requirements Specification (SRS) and Software Design Description (SDD). The documents are available at https://bit.ly/2C9VBro.

During the course, 10 different groups performed OOAD for the selected case. We selected this group (consists of six junior (third-year) computer engineering students) as the members have experience in the sector, are familiar with microservices and their documents had the highest grades.

For the analysis, they used use-cases to depict the functional requirements. First, they designed a use-case diagram. Then, they created use-case descriptions including use-case scenarios. For the design, they utilized a class diagram for the logical viewpoint, sequence diagrams for the sequential viewpoint, and a component diagram for the interface viewpoint. They also designed a tree diagram to show the relational database. All diagrams were designed as low level and represent the design in detail.

The group members easily conducted the analysis process. However, they had some difficulties during the design process when they followed a traditional OO approach to implement an MS-based solution. They found OO design not helping to break the problem into microservices. When they consulted us, we recommended they use a component diagram to show the internal and external interfaces of the system. Using the component diagram, they identified the external interfaces which show the communication between external systems and

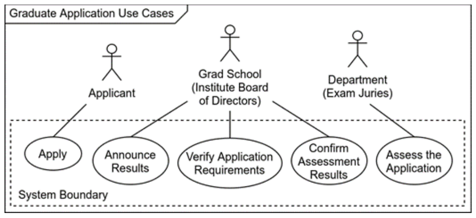
Graduate Application Use Cases
Applicant
Grad School (Institute Board of Directors)
Department (Exam Juries)
Apply
Announce Results
Verify Application Requirements
Confirm Assessment Results
Assess the Application
System Boundary

Fig. 6. Use case diagram for graduate application

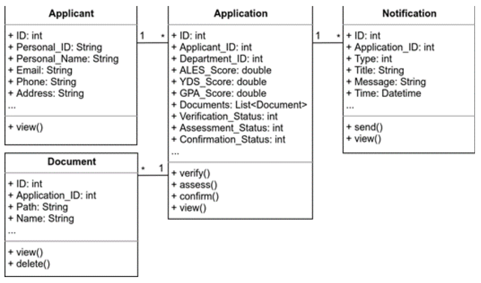

Fig. 7. Class diagram for graduate application

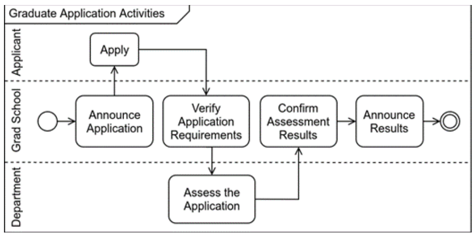

Fig. 8. Activity diagram for graduate application

their solution. However, they were confused about identifying internal interfaces. They could not decompose the system into microservices and thus, they could not show the communication between microservices. A common question from group members was "How can we decompose the system to microservices based on our class diagram?". Also, they asked how to show the extracted classes in the component diagram.

In this case study, students initially used a class diagram to design the system. Then, they attempted to design a component diagram that is compatible with the class diagram. In other words, they tried to utilize both diagrams to design the system. However, this approach resulted in three problems: (1) the initial class diagram could not describe the organization and wiring of the internal system components, (2) class diagrams caused confusion for the microservice communication, (3) eventually the class diagram became meaningless when the component diagram viewpoint is used. If the sequence had been changed, they would have been more successful to design a component diagram and show the internal interfaces. Class-based decomposition caused more confusion than usage.

As a result, OO design was not able to provide a strategy for developing an MS-based solution for the given case. The group tended to develop a model-based decomposition strategy and the design has followed the Model-View-Controller (MVC) pattern. As a result, the group implemented the project based on the MVC design pattern. They deployed their solution to the cloud and managed to communicate with external services. However, their solution became a monolithic system.

V. Discussion

In this study, we aimed to answer if OO or EO analysis approaches provide a better strategy for developing an MS-based solution. Accordingly, we will compare and evaluate both approaches for their success in meeting the important characteristics of an MSbA as follow:

Loose coupling: In MSbA, the degree of interdependence between the modules (i.e., the subdomains) is expected to be as low as possible. Thus, the application as a domain should be decomposed in a way that each business process is handled by a different service. OOAD provided loose coupling at the class level. However, it did not guarantee the separation of subdomains or business capabilities. On the other hand, EOAD addressed this issue and ensured the separation of different sub-processes, such as application, verification, evaluation, and notification. Thus, we easily managed to identify loosely coupled microservices after applying EOAD.

Cohesion: Similarly, OOAD focuses on the cohesion between classes to adhere to object-oriented data modeling. Although the functionally related elements have related attributes and methods, the class-based decomposition does not help to keep all functionally related processes together. On the other hand, since the separation is based on the business processes and bounded context, the EOAD helps identify highly cohesive microservices by keeping the functionally relevant elements and flows together. As an example, the application microservice manages all application-related events and activities.

Isolation: An MSbA requires proper isolation to be able to test, maintain, and deploy each microservice without affecting any others. Isolation is also a prerequisite for fault tolerance and autonomy. However, it is not possible to create a fully isolated class structure based on the class decomposition of OOAD methods because it typically points to a shared data structure. On the other hand, EOAD promotes a distributed database system (including event logs) where each microservice has its own data collection. In this way, the dependency on shared data is resolved.

Autonomy: In addition to having an isolated database, each microservice is expected to be able to operate as an autonomous service that takes full responsibility for a single business capacity. Here, full responsibility includes presentation, API, and business logic in addition to data storage. In EOAD, since each microservice is designed as an independent application, it can be discovered and run by users or other microservices, individually. On the other hand, OOAD requires all systems to be up and running even if only one presentation, API request, or business logic is needed.

Composability: The existing microservices should be decomposable to perform more complex or new business processes. In our example, application, verification, evaluation, and notification are sub-processes that each is handled by an individual microservice. However, they cooperate and coordinate with each other to manage the whole process of the graduate student application. Similarly, they are designed in a way that they can be integrated with external microservices to perform other possible use-cases. These compositions do not require additional changes on the existing microservices because they all have a uniform interface complying with SOA principles and REST architectural style. Here, the key difference between EOAD and OOAD is that EOAD adopts SOA which requires stateless services, while OOAD adopts OO which fits well in a stateful environment, by its nature.

Small bounded context: In EOAD, the bounded context is identified by the events. The event draws the boundaries of the context. On the other hand, in OOAD, class diagrams are a way to decompose the bounded context. The boundaries are decomposed as classes such as applicant, application, document, and notification. In our use case, both approaches were successful in determining a well-bounded context for business logic and data. However, as EOAD defines boundaries based on events it naturally corresponds to Microservices to be developed.

Asynchronous communication: In EOAD, an event triggers a microservice and upon the completion of it generates another event. Since the automated process is based on events and each bounded context generates an event and is triggered by an event, asynchronous communication is implemented. We could easily implement asynchronous communication using message queues. On the other hand, in OOAD, communication is between the methods of classes that could not provide a strategy for asynchronous communication.

Single responsibility: Both EOAD and OOAD aim to decompose the system in a way that each component (class or subdomain) performs a single business process. EOAD decomposes the system based on event boundaries where each event represents a single business process such as application, verification, evaluation, and notification. On the other hand, OOAD depicts the decomposition of the system with respect to classes. Similarly, each class performs a single business process of a class. To illustrate, the applicant class performs business processes related to an applicant. However, the single responsibility of the class does not naturally correspond to Microservices to be developed.

Scalability: Each business process should be scalable in MSbAs. For example, in our case, the application service is scaled up on load (i.e., during the application period). However, when the application period is over, it can be scaled down. EOAD provides high scalability since microservices for each business process are deployed independently. On the other hand, OOAD decomposes the problem based on classes. Thus, OOAD does not provide a viewpoint for high scalability for each business process. For example, only the classes relevant to the application process need to be scaled up during the application period. However, OOAD cannot provide such a decomposition that allows partial scaling.

Fault tolerance: Loose coupling, isolation, and autonomy of microservices provide fault tolerance in comparison to a monolith application. Each microservice should continue to operate without the existence of any other microservice. To illustrate, the application microservice should continue to operate even if the notification microservice fails. Each application should be stored in a structure such as a queue so that the notification service can manage these applications without any loss. The nature of EOAD provides fault-tolerant microservices. Each microservice is designed to manage a sub-process. If any microservice fails, other sub-processes will continue to operate. On the other hand, if a system is designed on a class-based decomposition which is less tolerant of system failures as the classes used by multiple sub-processes cause an inter-process dependency over the shared methods and data attributes.

One of the validity threads of our study is that third-year students participated in OOAD. Although we chose the best group and they have some experience with microservices, one can think that they are relatively inexperienced. This might be true but they are not the only group that applied OOAD with MSbAs. We involved them as a second group in OOAD to prevent potential bias of the authors. The second validity thread can be that our study is based on one case study and real-life industrial cases can be more complicated with many business processes and parallel activities. In this study, we aimed to show that the applicability of EOAD in MSbAs, and this approach can be validated in more complicated industrial cases.

VI. Conclusion

The traditional monolithic applications are changing to small, isolated, loosely-coupled applications that work in cohesion. These applications are called microservices and they can be deployed, scaled, and tested independently. MSbA has become a popular and efficient way for the development of software. However, the design and analysis of microservices may not be clear for the companies as there is not a de facto model proposed for the analysis and design of microservices.

The industry is still using the traditional approaches [4], [18]–[20] for the analysis and design of MS-based systems but these approaches do not cope with the demands of the new generation of systems. In this study, we explored a novel approach for the analysis and design of microservices: Event-Oriented analysis and design (EOAD). We have observed that an event-based modeling approach is highly useful for analyzing and designing MS-based solutions.

We applied EO and OO analysis methods to analyze a problem to develop a microservice architecture-based solution. We tried to answer the research question: "How successful are OOAD and EOAD in meeting the important characteristics required by an MSbA?" and conducted a case study to compare the approaches.

MSbA differs from traditional monolithic applications in many ways. The structural decompositions required by microservices are quite different from those of OOAD viewpoints. Thus, we discussed the success of EOAD and OOAD in terms of meeting these characteristics. Overall, we observed that OOAD, by its nature, does not have useful viewpoints to analyze and design an MS-based solution that meets the important characteristics of an MSbA. On the other hand, we observe that EOAD can be useful for this task.

This research presents the applicability of EOAD for MSbAs. However, there is still a need to develop alternative approaches to be used in modeling MS-based systems.

Acknowledgment

We thank Baran Ates, Efe Can Cevher, Kaan Algan, Kemal Selcuk Kaplan Zekihan Azman for contributing the case study.

References

[1] N. Alshuqayran, N. Ali, and R. Evans, "A Systematic Mapping Study in Microservice Architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, Nov. 2016, pp. 44–51. doi: 10.1109/SOCA.2016.15.

[2] J. Thönes, "Microservices," *IEEE Software*, vol. 32, no. 1, Art. no. 1, Jan. 2015, doi: 10.1109/MS.2015.11.

[3] J. Bonér, *Reactive Microservices Architecture*. O'Reilly Media, Inc., 2016.

[4] B. Bilgin, H. Unlu, and O. Demirörs, "Analysis and Design of Microservices: Results from Turkey," in *2020 Turkish National Software Engineering Symposium (UYMS)*, Oct. 2020, pp. 1–6. doi: 10.1109/UYMS50627.2020.9247022.

[5] A. Brandolini, *Event Storming*. Leanpub, 2019. [Online]. Available: https://leanpub.com/introducing_eventstorming

[6] "EventStorming," *EventStorming*, Mar. 18, 2020. https://www.eventstorming.com/ (accessed Mar. 18, 2020).

[7] Z. Stojanovic, A. Dahanayake, and H. Sol, "Modeling and design of service-oriented architecture," in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, Oct. 2004, vol. 5, pp. 4147–4152 vol.5. doi: 10.1109/ICSMC.2004.1401181.

[8] S. J. Andriole, "The death of big software," *Commun. ACM*, vol. 60, no. 12, Art. no. 12, Nov. 2017, doi: 10.1145/3152722.

[9] N. Dragoni *et al.*, "Microservices: Yesterday, Today, and Tomorrow," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds. Cham: Springer International Publishing, 2017, pp. 195–216. doi: 10.1007/978-3-319-67425-4_12.

[10] S. Li *et al.*, "A dataflow-driven approach to identifying microservices from monolithic applications," *Journal of Systems and Software*, vol. 157, p. 110380, Nov. 2019, doi: 10.1016/j.jss.2019.07.008.

[11] M. Kamimura, K. Yano, T. Hatano, and A. Matsuo, "Extracting Candidates of Microservices from Monolithic Application Code," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, Dec. 2018, pp. 571–580. doi: 10.1109/APSEC.2018.00072.

[12] O. Al-Debagy and P. Martinek, "A New Decomposition Method for Designing Microservices," *Periodica Polytechnica Electrical Engineering and Computer Science*, vol. 63, no. 4, Art. no. 4, Jun. 2019, doi: 10.3311/PPee.13925.

[13] S.-P. Ma, C.-Y. Fan, Y. Chuang, I.-H. Liu, and C.-W. Lan, "Graph-based and scenario-driven microservice analysis, retrieval, and testing," *Future Generation Computer Systems*, vol. 100, pp. 724–735, Nov. 2019, doi: 10.1016/j.future.2019.05.048.

[14] N. Santos *et al.*, "A logical architecture design method for microservices architectures," in *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, Paris, France, Sep. 2019, pp. 145–151. doi: 10.1145/3344948.3344991.

[15] L. Baresi, M. Garriga, and A. De Renzis, "Microservices Identification Through Interface Analysis," in *Service-Oriented and Cloud Computing*, Cham, 2017, pp. 19–33. doi: 10.1007/978-3-319-67262-5_2.

[16] J. Bonér, *Reactive Microsystems*. O'Reilly Media, Inc., 2017.

[17] A.-W. Scheer, O. Thomas, and O. Adam, "Process Modeling Using Event-Driven Process Chains," *Process-aware information systems*, no. 119, Art. no. 119, 2005.

[18] D. Cooke, A. Gates, E. Demirörs, O. Demirörs, M. M. Tanik, and B. Krämer, "Languages for the specification of software," *Journal of Systems and Software*, vol. 32, no. 3, Art. no. 3, Mar. 1996, doi: 10.1016/0164-1212(95)00071-2.

[19] D. Akdur, B. Say, and O. Demirörs, "Modeling cultures of the embedded software industry: feedback from the field," *Softw Syst Model*, Jun. 2020, doi: 10.1007/s10270-020-00810-9.

[20] D. Akdur, V. Garousi, and O. Demirörs, "A survey on modeling and model-driven engineering practices in the embedded software industry," *Journal of Systems Architecture*, vol. 91, pp. 62–82, Nov. 2018, doi: 10.1016/j.sysarc.2018.09.007.