MICROARC: AN ANALYSIS AND DESIGN METHOD FOR MICROSERVICE BASED SYSTEMS

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in Computer Engineering

by
Ali YILDIZ

December 2024
İZMİR

We approve the thesis of **Ali YILDIZ**

Examining Committee Members:

Prof. Dr. Onur DEMİRÖRS
Department of Computer Engineering, Izmir Institute of Technology

Prof. Dr. Tuğkan TUĞLULAR
Department of Computer Engineering, Izmir Institute of Technology

Prof. Dr. Oğuz DİKENELLİ
Department of Computer Engineering, Ege University

Assoc. Prof. Dr. Rıza Cenk ERDUR
Department of Computer Engineering, Ege University

Asst. Prof. Dr. Emrah İNAN
Department of Computer Engineering, Izmir Institute of Technology

12 December 2024

Prof. Dr. Onur DEMİRÖRS
Supervisor, Department of Computer
Engineering, Izmir Institute of Technology

Prof. Dr. Onur DEMİRÖRS
Head of Department of Computer
Engineering, Izmir Institute of Technology

Prof. Dr. Mehtap EANES
Dean of the Graduate School of
Engineering and Sciences

ACKNOWLEDGMENTS

First of all, I express my sincere gratitude to my supervisor, Prof. Dr. Onur Demirörs. I am thankful to him for his continuous guidance, support, patience, and encouragement throughout my study.

I am grateful to my committee members, Prof. Dr. Tuğkan Tuğlular and Prof. Dr. Oğuz Dikenelli, for their ideas and support throughout the thesis study.

I would like to thank Assoc. Prof. Dr. Rıza Cenk Erdur and Asst. Prof. Dr. Emrah İnan for their valuable advice.

ABSTRACT

MICROARC: AN ANALYSIS AND DESIGN METHOD FOR MICROSERVICE BASED SYSTEMS

The rapidly developing internet infrastructure has enabled software applications to leverage almost unlimited and scalable resources. Microservice-based architecture (MSbA) has emerged as a solution to harness the benefits of a distributed cloud-based infrastructure.

MSbA is becoming a popular approach for creating distributed software systems, emphasizing the design and development of maintainable, easily scalable, and highly available systems. It comprises loosely coupled and highly cohesive independent services, known as Microservices, which communicate over a network to execute high-level processes. Microservices can be updated or deployed independently, and interruptions in their operations do not affect the entire system. They can communicate using technology-independent protocols, such as HTTP and REST, and developers can choose different development languages or platforms for each microservice based on what best suits its functionality.

Event-driven architecture is a powerful approach for addressing challenges in distributed systems, such as scalability, distributed data, and sharing of data at scale. In event-driven Microservices architecture, decoupled services interact by responding to events, and event streams facilitate data sharing between them.

Despite these advantages, there is no de facto method for the analysis and design of MSbA. Organizations often face difficulties in developing microservice-based systems, owing to the lack of well-defined methodologies for analysis and design. This thesis proposed an event-oriented analysis and design method for MSbA. The method comprises three main components; the processes to be employed in the analysis and design phases, the modeling notations utilized in these processes, and the heuristics that support the analysis and design phases.

ÖZET

MICROARC: MİKROSERVİS TABANLI SİSTEMLER İÇİN BİR ANALİZ VE TASARIM METODU

Hızla gelişen İnternet altyapısı, yazılım uygulamalarının neredeyse sınırsız ve ölçeklenebilir kaynaklardan yararlanmasını sağlamıştır. Mikroservis tabanlı mimari (MSbA), dağıtılmış bulut tabanlı bir altyapının faydalarından yararlanmak için bir çözüm olarak ortaya çıkmıştır.

MSbA, bakımı kolay, kolayca ölçeklenebilir ve yüksek oranda kullanılabilir sistemlerin tasarımını ve geliştirilmesini sağlaması nedeniyle dağıtılmış yazılım sistemleri oluşturmak için popüler bir yaklaşım haline gelmektedir. Mikroservisler , üst düzey süreçleri yürütmek için bir ağ üzerinden iletişim kuran, gevşek bir şekilde bağlanmış ve son derece uyumlu bağımsız hizmetlerden oluşur. Mikroservisler bağımsız olarak güncellenebilir ve dağıtılabilir ve operasyonlarındaki kesintiler tüm sistemi etkilemez. HTTP ve REST gibi teknolojiden bağımsız protokolleri kullanarak iletişim kurabilirler ve geliştiriciler, her bir mikroservis için işlevselliğine en uygun olan farklı geliştirme dilleri veya platformları seçebilirler.

Olay odaklı mimari, ölçeklenebilirlik, dağıtılmış veri ve ölçeklenmiş veri paylaşımı gibi dağıtılmış sistemlerdeki zorlukların üstesinden gelmek için güçlü bir yaklaşımdır. Olay odaklı Mikroservis mimarisinde, ayrıştırılmış hizmetler olaylara yanıt vererek etkileşime girer ve olay akışları aralarında veri paylaşımını kolaylaştırır.

Bu avantajlara rağmen, bu mimarideki sistemlerin analizi ve tasarımı için yaygın kullanılan bir yöntem yoktur. Kuruluşlar, analiz ve tasarım için iyi tanımlanmış metodolojilerin olmaması nedeniyle genellikle mikroservis tabanlı sistemler geliştirmede zorluklarla karşılaşırlar. Bu tez, mikroservis mimarisinde yazılım geliştirme için olay odaklı bir analiz ve tasarım yöntemi önermiştir. Yöntem üç ana bileşenden oluşur; analiz ve tasarım aşamalarında kullanılacak süreçler, bu süreçlerde kullanılan modelleme notasyonları ve analiz ve tasarım aşamalarını destekleyen sezgisel yöntemlerdir.

to my beloved family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

API	Application Programming Interface
BMC	Business Model Canvas
CD	Continuous Deployment
CI	Continuous Integration
DB	Data Base
DDD	Domain Driven Design
EDA	Event Driven Architecture
EPC	Event-Driven Process Chain
EOAD	Event Oriented Analysis and Design
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
HTTP	Hyper Text Transfer Protocol
MEFD	Microservice Event Flow Diagram
MSbA	MicroService Based Architecture
OOAD	Object Oriented Analysis and Design
REST	Representational State Transfer
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UML	Unified Modeling Language

CHAPTER 1

INTRODUCTION

Currently, software applications can leverage nearly unlimited and scalable resources, and cloud platforms are widely employed to provide these capabilities. The distribution of an application's business logic across different platforms is facilitated by the utilization of cloud platforms.[1] Microservice-based architecture serves as a solution to achieve this objective, where services operate with loose coupling and cohesion. In this architecture, services can be deployed independently, are scalable, and offer testability. Lightweight frameworks such as HTTP and REST are commonly utilized for establishing connections between services.[1,2]

Event-driven architecture handles business processes using the flow of messages through loosely coupled services. Each service belonging to a given domain or bounded context has a specific role and a limited responsibility within that domain. Each domain processes relevant data and communicates these changes to the other domains.

There are many benefits of using event-driven microservices.[3] Granularity: Services can be easily rewritten when business requirements change. Scalability: Services can be scaled up or down as required. Technological independence: Services use the most appropriate language and technology. Business requirement flexibility: Service ownership can be easily recognized because there are fewer cross-team dependencies. Loosely Coupling: Services are coupled with domain data. Continuous delivery support: It is easy to ship small and modular services. High testability: Services tend to have fewer dependencies than monoliths.

In this study, an event-driven analysis and design method, along with modeling approaches to support this method, is presented for building microservice-based architectural systems. The event-driven analysis and design method comprises three main components: the processes to be employed in the analysis and design phases, the modeling notations utilized in these processes, and the heuristics that support the analysis and design phases.

1.1. Background of the Problem

A microservice-based architecture is a new approach for building distributed software systems. They focused on designing and producing maintainable, scalable, and high-availability systems.[4] Services in a microservice architecture are independent components that use lightweight communication to achieve their goals. Microservices can communicate with each other using technology-independent protocols. Therefore, the selection of development languages and running platforms can differ for each microservice. These are independently updated or changed, and the failure of the microservice does not affect the entire system.

The use of microservice-based architecture is continuously increasing among software companies that want to build distributed, scalable, and highly available systems. A survey[5] conducted by the Software House in 2020 showed that companies prefer to use Microservice-based architecture (MSbA) for applications that require high performance and scalability (70% and 84%, respectively). Moreover, a survey[6] conducted in Turkey in 2020 showed that microservice-based architecture uses different domains, such as automation, ERP, e-commerce, web-based, logistics, finance, mobility, telecommunication, and public service applications.

Organizations face many difficulties in developing microservice-based systems owing to the lack of well-defined methodologies for their analysis and design. Temporary methods, mostly Object-Oriented Analysis and Design (OOAD), are used.[6] However, the structural decomposition of microservice-based systems differs significantly from that of OOAD. Additionally, traditional analysis and design techniques have been found to be ineffective for microservice architectures.[7] Alternative approaches, such as Event Storming, do not have systematic implementation processes, model notations used in different stages, or software tools that support the methodology.[8]

1.2. Purpose of the Study

This study aims to develop an efficient method and supporting tool for analysis and design of event-oriented microservice systems. The proposed method presents an analysis and design approach for developing microservice based systems.

Software development starts with eliciting requirements and then establishing an architecture that defines tiers, components of the system to be built, and interactions between them. The main objective of the requirements elicitation phase is to elicit functional and quality requirements, business rules, and define business processes. MicroArc method enables event-based modeling of business processes which is crucial for event-oriented microservices systems. The method helps analysts to define boundaries of microservices, events that trigger microservices, or are initiated by microservices. In addition, it enables define event flows between microservices.

The goals of this study are to help software organizations that want to develop event oriented microservice systems. Using MicroArc method and tool, a software company can model a system with different perspectives that are connected between them, and define events and microservices from these modes.

1.3. Research Strategy

The design-science approach used in this study.[9] Offermann[10] structured three main phases for design-science process; problem identification, solution design, and evaluation (Figure 1.1).

In problem identification phase; a literature review was conducted to find out which modeling notations already existed and which implementation processes are mostly used for analysis and design of Microservice Based Systems. Based on literature research, we found that there is no de-facto method for analysis and design of Microservice Based Systems, however Object-Oriented Analysis and Design (OOAD) approach mostly is used.[6] An exploratory case study conducted to find "How successful are Object Oriented Analysis and Design (OOAD) and Event Oriented Analysis and Design (EOAD) in meeting the key characteristics required by a Microservices based Architecture?". The results showed that EOAD meets most key characteristics required by a MSbA than OOAD.[7] Moreover, modelling and implementation processes requirements were identified.

In solution design phase; modeling notations and implementation processes are defined based on requirements that were identified in problem identification phase, and first version of MicroArc method has been formed. A second exploratory case study was conducted in order to identify needs for effective use of MicroArc method. Based on the case result, second version of MicroArc method has been formed. Additionally, supporting tool requirements for the analysis and design of MSbA were identified, and the MicroArc tool was developed.

In evaluation phase; after the finalization of the MicroArc model and tool, an explanatory case study was conducted to find out, "How effective is the MicroArc method for events and microservices identification?" by comparing two studies that analyzed and designed the same application.

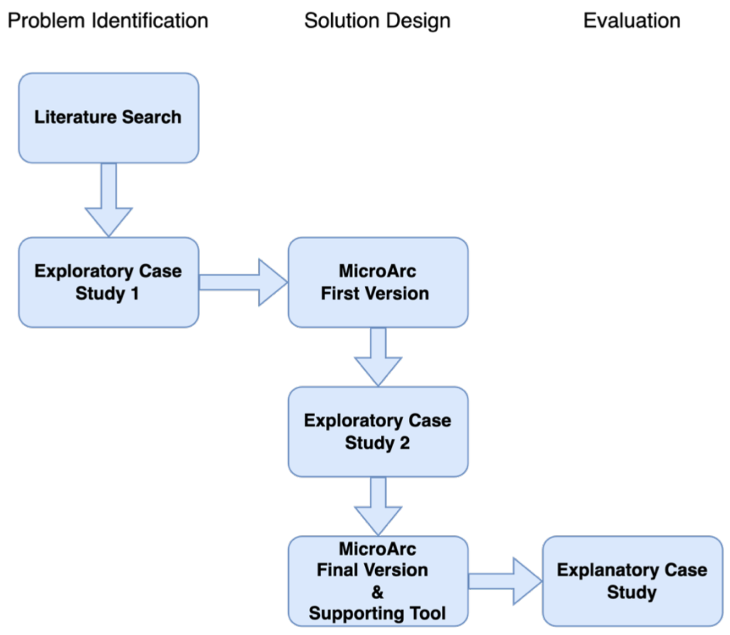


Figure 1.1. Research Strategy process

1.4. Contribution and Significance of the Study

The major contribution of this study is a method and tool, called MicroArc, which is used to analyze and design microservice based systems. MicroArc tool consists of a modeling environment and modeling notations. By using the tool, an analyst can model a software system from three different perspectives; BMC (Business Model Canvas) provides to define key activities that show domains and ubiquitous languages, EPC (Event-driven Process Chain) provides to model business processes in event-oriented way, and identify events and microservices, MEFD (Microservice Event Flow Diagram) provides to show event flows between microservices. Each model is connected to each other, so any changes to a model can be reflected in other models.

MicroArc method is a novel approach to analyze and design event-oriented microservice systems. Additionally, MicroArc tool is the first tool that establishes connections between models to reflect changes in a model to other models.

1.5. Organization of the Thesis

The thesis is organized as follows: Chapter 1 briefly summarized the background of the problem, purpose of the study, research strategy, contribution and significance of the study. Chapter 2 gives a discussion about on microservice architecture and related researches on analysis and design of microservices. Chapter 3 specifies the MicroArc method and supporting tool. Chapter 4 describes the case studies which are conducted to explore and validate the proposed method. Chapter 5 provides an overall discussion of the contributions, limitations and future work regarding with this thesis.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

This chapter consists of seven sections. The first describes Microservices Based Architecture. The second section describes Event Driven Architecture. The third section describes Service Oriented Architecture (SOA). The fourth section describes Domain Driven Design (DDD). The fifth section describes Event Storming. The sixth section describes related studies regarding to developing microservices based architecture. Finally, the seventh section gives a brief discussion of the state of the art.

2.1. Microservices Based Architecture

A microservice-based architecture is a new approach for building distributed software systems. They focused on designing and producing maintainable, scalable, and high-availability systems.[4] Services in a microservice architecture are independent components that use lightweight communication to achieve their goals. Microservices can communicate with each other using technology-independent protocols such as HTTP and REST. Therefore, the selection of development languages and running platforms may differ for each microservice. Microservices are updated or changed independently, and failure of a microservice does not affect the entire system. Based on this definition a microservice can be independently deployable, scalable, testable, and has a single responsibility. Microservice based architecture is depicted in Figure 2.1.

The main characteristics of microservices are given below:

- **Independently deployable:** a microservice has low coupling between them so it can be run on different environments and can be deployable independently without affecting others.
- **Independently scalable:** Low coupling between microservices enable them easily scale up or down when more request and resource needed.

- **Independently testable: a microservice can be tested independently without need any interaction with other microservices. It makes preparing test cases and testing simpler and smoother.**
- **Single responsibility**: it means that microservice should have one, only one reason to change or to be replaced.[11]

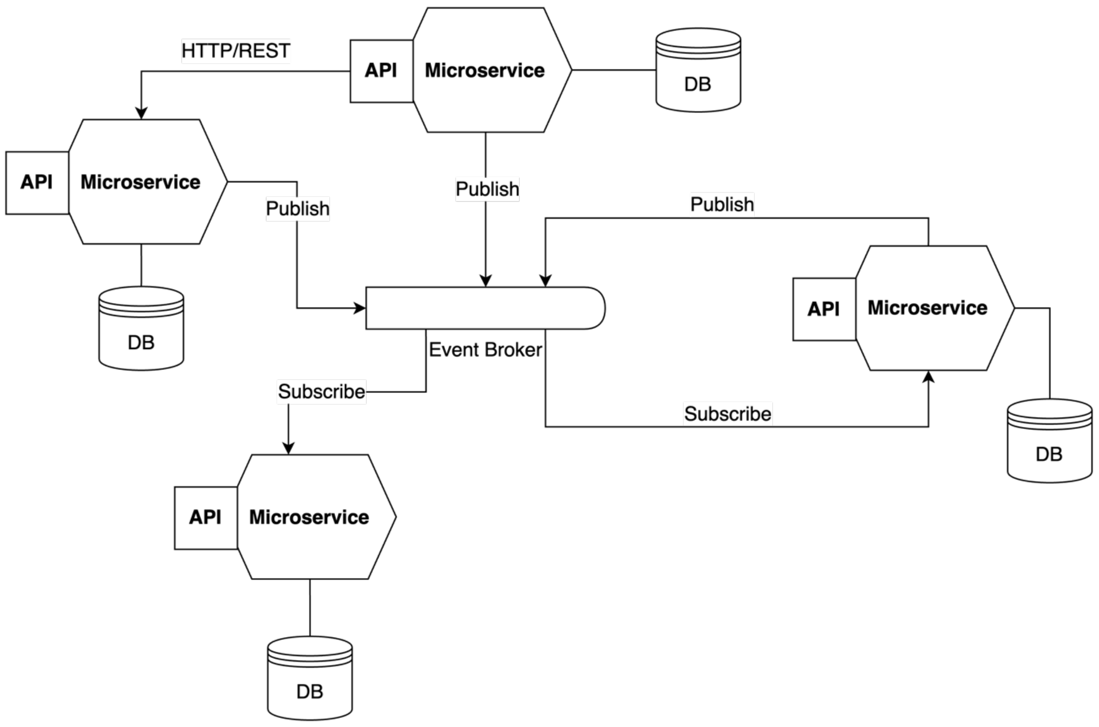


Figure 2.1. Microservice Based Architecture

Microservices Based Architectures have the following characteristics:

- **Componentization via Services:** It is desirable to create software systems through components. In many software platforms this is provided by libraries. Using libraries as components in microservice architectures creates high cohesion since libraries are directly accessible from within the program. While libraries are linked into a program and called using in-memory function calls, services are out of process components who communicate via networks. It becomes services independently deployable and it is one main reason to use services as components rather than libraries. This approach also provides well defined interfaces.

- **Organized around Business Capabilities**: Many software development organizations have a horizontal distribution of responsibilities between project teams. Each team develops different part of software that is under their responsibility such as user interface, business logic, database. Any change can lead to many cross-team communication and take time. Services are organized around business capability in microservice architecture, and project teams are responsible for each service. Teams are cross-functional with all skills necessary to develop a service such as project management, database, etc. This reduces cross-team communications and enables rapid software development.
- **Products not Projects:** In a project model, aim is to deliver working code. When a project completed software development team delivers software to the maintenance team and starts to another project. The goal in microservice approach is to develop a software that assist its user to enhance the business capability rather than looking at the software as a set of functionalities to be completed. A software development team takes full responsibility for the software in production. Same team maintains and updates the software through its full lifetime. This establishes a long-term relationship with customers to develop their necessary needs and increases customer satisfactions.
- **Smart endpoints and dump pipes:** Most products or approaches include advanced features in the communication mechanism when building a communication structure between different processes. Enterprise Service Bus (ESB) used by Service Oriented Architecture (SOA) is an example for this. ESB products include advanced capabilities for message routing, choreography, transformation, and applying business rules. This approach makes communication complex and difficult to manage it. In the microservice architecture, communication is done with simple protocols, each service receives a request, applies its business capability, and produces a response. This speeds up development and simplifies maintenance of an application.
- **Decentralized Governance:** In most software development organizations; a defined set of software development platforms and databases, etc. are commonly used in all software development projects. It is difficult and uncommon to use different structures in different parts of an application. In microservice architecture, each team is responsible for the service to be developed and decides

the structures to be used. It allows to use most appropriate technologies for a service’s business capability.

- **Decentralized Data Management:** Using a shared database in a software application has many benefits in terms of database creation, management and cost. However, in case of inaccessibility to the database, whole system becomes inoperable. In microservice architecture, every service has its own database. It allows the use of appropriate databases according to business capabilities of services and prevents become unusable of entire system when a problem occurs.
- **Infrastructure Automation:** Infrastructure automation reduces the operational complexity of building, integration, testing, deploying, and operating of software applications, also it provides short development life cycle. In microservice architecture, infrastructure automation is important for short development cycle, therefore infrastructure automation approaches such as Continuous Integration (CI) and Continuous Deployment (CD) are widely used.
- **Design for failure:** In microservice architecture, services work together to perform business capabilities and they communicate over network. Since service can fail at any time, monitoring status of services, detecting the failures quickly, and automatically compensate and restoring service, take into account. Thus, system reliability is established.
- **Evolutionary Design:** The fundamental principle of an evolutionary design is change. Microservices are designed in such a way that allows them to be updated, improved, and extended over time. The system is broken into components that are independently replaceable and upgradable, to perform evolutionary design.

2.2. Event Driven Architecture

An event is a well-defined record that contains all the information about what happened in the business.[12] It is typically based on entities or on relationships between them. Events are named in the past tense. For example, an e-commerce order may result in a “payment received” event. Event is immutable, meaning that once published, it cannot be changed.

Event Driven Architecture (EDA) is an architectural concept in which services of autonomous software components interact via event notifications.[13] EDA has three main components; event producer, event consumer, and event router (Figure 2.2). Event producer generates events in response to specific occurrence or changes inside the system. Event consumer listens for specific events and responds according to event data. Event router orchestrates events and routes events from producers to interested consumers.

Decoupling and asynchronous events are the key concepts related to EDA. Both make systems resilient and flexible. In decoupling, components in systems exchange information through events, and there is not directly communication between components. Decoupling enables developers to modify individual components and scale down or up them without impacting the whole system. It also improves the maintainability and prevents the occurrences of cascading failures. Asynchronous events allow components to communicate without being locked up and waiting for instant response or acknowledgement from other components.

An event-driven microservices architecture is a design paradigm that integrates event-driven architecture with microservices, yielding solutions composed of loosely linked services that interact via asynchronous events. So, a microservice can be developed, tested, deployed, upgraded and scaled independently of other microservices.

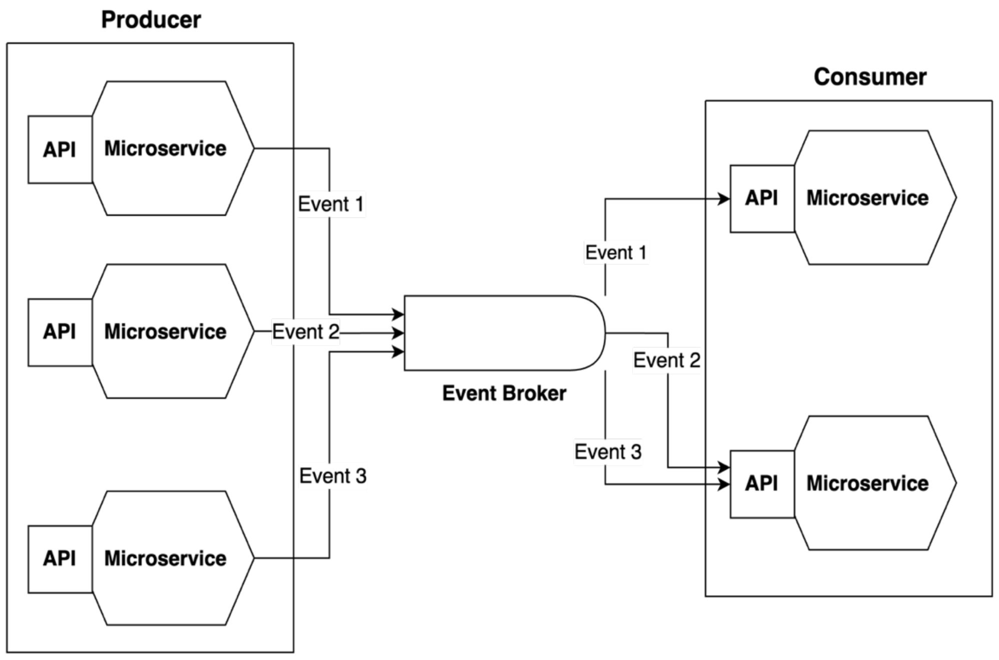


Figure 2.2. Event-Driven Architecture

2.3. SOA and Microservices

Service Oriented Architecture (SOA) is a way of designing software that uses software components called services to create business applications, each accessible through standard interfaces and messaging protocols.[14] In SOA, independent software components provide services to end-user applications or other services. The three primary participants in SOA are the service registry, service provider, and service consumer. The interactions between them involve publish, find, and bind operations.[14] The service provider defines a service description of the service and publishes it to a service registry to make a service description that is published and made discoverable. The service consumer discovers the service description on the service registry by using the find operation. Using the bind operation, the service consumer invokes the service from the service provider (Figure 2.3).

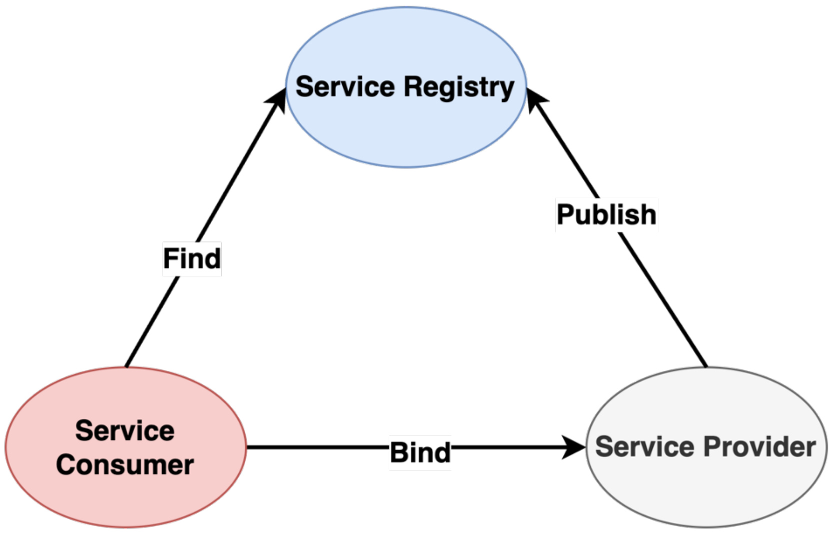


Figure 2.3. SOA Publish, Find, Bind paradigm

Microservice Based Architecture (MSbA) which aims to create, small, isolated, loosely-coupled applications that work in cohesion, is considered as an evolution of SOA. The goal of MSbA and SOA is to decompose monolithic applications to modular services.

However, SOA concentrates on enterprise-wide integration with centralized management, MSbA prioritize autonomous, single-purpose services with decentralized governance.

Main differences between SOA and Microservices are summarized below:

- **Communication:** SOA uses ESB (Enterprise Service Bus) communication channel to manage and coordinate services. The ESB can become a risk of a single point of failure for the whole system. Microservices communicate using API in order to avoid this risk.
- **Interoperability:** SOA uses multiple message protocols like SOAP (Simple Object Access Protocol), and AMQP (Advanced Messaging Queuing Protocol). Microservices use lightweight messaging protocols like HTTP/REST, JMS (Java Messaging Service), and Kafka.
- **Governance:** In SOA, component reusability is desired, and services share resources. This requires the implementation of common data governance standards for all services. Instead, in MSbA does not implement consistent data governance to provide independence for all microservices.
- **Granularity:** In SOA, services can range from small specialized to relatively large enterprise-wide. Microservices, on the other hand, are smaller, highly specialized services, each of which is designed to serve a specific purpose.
- **Storage:** In SOA, services share a single data storage layer. However, this creates dependencies across services. MSbA provides independent data storage for each service as required to maintain loose coupling among microservices.

2.4. Domain Driven Design and Microservices

Domain-Driven Design is a methodology for developing complex software systems that emphasizes the core domain. It encourages innovative collaboration between domain experts and software developers to create models.[15] DDD is typically considered from two perspectives; Strategic design, and Tactical design.

The goal of Strategic Design is to define a software system's general design and structure in a way that is consistent with the problem domain. It tackles high-level problems including how to divide the system into manageable parts, how to clearly define the boundaries between different components, and how to arrange domain concepts. Strategic design involves employing a ubiquitous language throughout the organization, dividing the system into distinct bounded contexts, and clearly defining the relationships between these contexts. This approach ensures that the entire team or organization is in alignment with the overall system structure and design. Main key concepts of Strategic Design are Bounded Context and Ubiquitous Language.

- **Bounded Context:** It is a conceptual boundary within a problem domain where it has a domain model and ubiquitous language. It allows to develop specific models for each context and sets clear boundaries to prevent confusion and inconsistency.
- **Ubiquitous Language:** It is a common language that all stakeholders use to define the domain model. It helps create a common understanding of domain concepts and requirements among technical and non-technical team members.

The goal of Tactical Design is to structure and organize the domain model within a software system. It helps clearly define rules and processes of the software system. It focuses on building and designing individual components in bounded context. Some key concepts of Tactical Design are: Entities, Value Objects, Services, Domain Events, Aggregates.

- **Entities**: An entity is an object that have a unique identity. It is mutable over time. Customers and Products in an e-shopping application would be entities.
- **Value Objects**: The value object has no identity, but it is defined by only by the values of its attributes. It is immutable. Address and Phone Number in an e-shopping application would be value objects.
- **Services**: a service is an object that implements some logic without holding any state. In DDD, domain services encapsulate domain logic, wheras application services provide technical functionality such as authorization.
- **Domain Events**: They are important events that occur within the domain, and they are used the notify other parts of the system.

- **Aggregates**: An aggregate is an encapsulation of entities and value objects which always need to be consistent, and are treated as a single of work. It helps to maintain consistency and integrity of the objects. In an e-shopping application, an aggregate would be an order that contains products, supliers, and should be modified together.

Although Tactical Design is not exclusively linked to the object-oriented paradigm, it leverages the benefits offered by the Objet Oriented (OO) approach, such as encapsulating data and behavior into objects. Entities, Value Objects, and Aggregates can correspond to classes in OO.

MSbA can have several benefits when applying the DDD principles. Bounded context defines the boundaries that each boundary has a particular model, and makes sure that each microservice has a unique model that is free from ambiguity. DDD establishes a common (ubiquitous) language in order to held proper communication between project stakeholders. Ubiquitous language helps microservices reflect business needs unambiguously. Since, a microservice has its own domain logic, data, and interactions, each microservice can be thought the implementation of a Bounded Context. However, a microservice is a bounded context, not every bounded context is a microservice. A Bounded Context defines boundaries of a service where it has a domain model that is isolated and consistent inside a bounded environment. One of key features of a microservice is that it has a single responsibility. A Bounded Context may have too many responsibilities to map it to a microservice. For example, in the e-commerce system, shipping can be identified as a Bounded Context. Shipping will most likely have multiple responsibilities such as scheduling, packaging, delivering, etc.

2.5. Event Storming

The Event Storming method was introduced by Brandolini[8]. It is a workshop-based method. In this workshop, participants from different areas work together to explore business domains by focusing on domain events generated in the context of a business process. Event Storming is commonly used by DDD practitioners, where it is based on DDD's main concepts such as domain events, commands, aggregates, etc. The implementation of Event Storming consists of five steps:

1. **Invitation of people:** The idea is to explore people's knowledge by meeting them in a meeting room. Ideally, six to eight people would attend the workshop. The group should consist of domain experts with knowledge of the domain and analysts who want to gather this domain knowledge.
2. **Providing modeling space:** Attendees use colored stickies to explore knowledge about the domain and stick it to the meeting room wall to see the whole.
3. **Exploring Domain Events:** A domain event is a trigger of an activity that happens in the domain. It can be the predecessor of the next event. An orange sticky note is used for an event. Events are placed on modeling space according to a timeline. Any duplicate events are removed or merged.
4. **Exploring sources of Domain Events:** If an event is a direct consequence of a user action, it is represented as a Command. A blue sticky note is used for a command. Other events can be a consequence of something happening in external systems that are represented using a purple sticky note. Other important information is represented by sticky notes with different colors.
5. **Searching for aggregates:** Commands and events are organized around an aggregate that represents a specific business concept.

An Event Storming workshop requires physical space with sticky notes, a pen, and large whiteboards. Although local teams can work well in this physical space, it can be difficult for remote teams. In addition, documenting all knowledge on whiteboards can be difficult. The involvement of all domain experts is desirable; however, participation can be more difficult.

2.6. Related Researches

Microservice analysis and design approaches can be separated in two; decomposition of microservices from an existing monolithic application and developing a new application in a microservice manner.[16-23]

A dataflow-driven approach for identifying microservices from monolithic applications proposes a dataflow-driven semi-automatic decomposition approach.[16] This approach introduces a four-step decomposition procedure: first, conduct the business requirements analysis to generate the use case and business logic specification; second, construct fine-grained Data Flow Diagrams (DFD); third, extract the dependencies between processes and datastores; and fourth, identify candidate microservices. This method relies heavily on the detailed DFDs at different levels. The quality of DFDs is critical for the decomposition results. Every process, external entity, data flow, and data storage for all the DFDs must be carefully reviewed.

Extracting Microservice Candidates from the Monolithic Application Code proposes a method that identifies the candidates of microservices from the source code using the software clustering algorithm SArF with the relation of "program groups" and "data" which is defined.[17]

A New Decomposition Method for Designing Microservices proposes an approach for decomposing a monolithic application into a microservice application by analyzing the application programming interface.[18] This methodology uses word embedding models to obtain word representations using operation names and a hierarchical clustering algorithm to group similar operation names to obtain suitable microservices.

Graph-based and scenario-driven microservice analysis, retrieval, and testing proposes an approach to the development of microservice-based systems, referred to as GSMART (Graph-based and Scenario-driven Microservice Analysis, Retrieval and Testing).[19] This enables the automatic generation of a "Service Dependency Graph (SDG)". The SDG is used to visualize and analyze the dependency relationships between microservices as well as between services and scenarios. It also enables automatic retrieval of test cases required for system changes to reduce the time and costs associated with regression testing. The GSMART approach mainly supports the Java-based spring-boot framework. Users are expected to follow defined guidelines to allow GSMART to retrieve the information required to produce and analyze SDGs. Moreover, GSMART uses Gherkin documents to specify test scenarios and service requirements.

A logical architecture design method for microservices architectures proposes a model-based approach for designing the logical view of an MSA, called 4SRS-MSLA.[20] This approach is based on modeling a business domain in UML use cases, thus deriving a UML component diagram for the domain and grouping the components into microservices.

Microservice Identification through Interface Analysis proposes a solution based on the semantic similarity of the foreseen or available functionality described in the OpenAPI specifications.[21] This approach relies on well-defined and described interfaces that provide meaningful names and follow programming-name conventions.

Service Cutter: A Systematic Approach to Service Decomposition proposes an approach to service decomposition based on 16 coupling criteria.[22] In the Service Cutter approach, coupling information is extracted from domain models and use cases and is represented as an undirected, weighted graph to find and score densely connected clusters. The resulting candidate service cuts promise to reduce coupling and promote cohesion within services.

Green Micro: Identifying Microservices from Use Cases in Greenfield Development proposes a method that identify microservices from business use cases by using a clustering algorithm on the combined similarity matrix.[23]

2.7. A Brief Discussion of the State of the Art

Microservices architecture is inspired by many architectural approaches. In Event Driven Architecture (EDA), services are autonomous components that interact via event notifications, as in event-driven microservices architecture that loosely linked microservices interact via asynchronous events.

In SOA, business applications are created as software components called services, each accessible through standard interfaces and message protocols. Microservices are small isolated, loosely-coupled applications that work in cohesion in MSbA which can be considered as an evolution of SOA. However, SOA concentrates on enterprise-wide integration with centralized management on services, MSbA prioritize autonomous, single-purpose services with decentralized governance.

DDD focuses on understanding domain, modelling it by exploring domain knowledge, defining sub-domains with bounded context in domain, employing a ubiquitous language within each bounded context to establish common understanding between stakeholder. Bounded context defines the boundaries that each boundary has a particular model, and makes sure that each microservice has a unique model that is free from ambiguity. Ubiquitous language helps microservices reflect business needs unambiguously.

Event Storming is a workshop for exploration of business domains by focusing on domain events that are essential for event-driven microservices architecture. A workshop requires physical space, and attending of domain experts to explore domain events. It can be challenging for remote participation and orchestrating many attendances.

In brief, the literature reveals that there is no de-facto method for analysis and design of Microservice-based architecture (MSbA). Our contribution in this study to provide an event-driven analysis and design method, called MicroArc, and a supporting tool for MSbA.

CHAPTER 3

ANALYSIS AND DESIGN METHOD FOR MICROSERVICES BASED SYSTEMS AND TOOL SUPPORT

In this chapter MicroArc method and tool for analysis and design of Microservices Based Systems is presented. The first section describes MicroArc method. The second section describes the MicroArc tool and its usage.

3.1. MicroArc: Event-Driven Analysis and Design Method

MicroArc is a method for analyzing and designing microservice-based systems that employs three modeling notations: Business Model Canvas (BMC), Event-driven Process Chain (EPC), and Microservices Event Flow Diagram (MEFD). The method also includes a guiding process that outlines the application of notations and a supporting tool that enables modeling and transition between models.

3.1.1. Business Model Canvas

Business Model Canvas (BMC) is utilized for modeling both new business ideas and documenting existing ones.[24] It encompasses nine building blocks within the business model design template and is categorized into four groups: Infrastructures, Offering, Customers, and Finances (Figure 3.1).

Infrastructures include followings:

- Key Activities: The vital activities necessary for the company's operations.
- Key Resources: The resources essential for performing key activities.
- Key Partnerships: The required suppliers and partners for business operations.

Offering includes followings:

- Value Propositions: The products or services offered by the company.

Customers include followings:

- Customer Segments: The target customers for the company's products or services.
- Channels: The delivery channels for the company's products or services.
- Customer Relationships: The types of relations the company maintains with customers to ensure business success.

Finances include followings:

- Cost Structure: The costs necessary to run the business.
- Revenue Streams: The income generated by serving products or services to customers

The BMC allows for a high-level analysis and provides overall information about the system that will be built. The software system is conceptualized as a business idea and evaluated from different perspectives. The offering perspective enables the identification of the benefits (value propositions) provided by software to customers. The customer perspective helps identify the users (customer segments) of the system, ways (channels) customers can reach it, and types of relationships (relationships) that can be established with customers. The infrastructure perspective enables the identification of key elements, such as activities, resources, and partnerships that are necessary to operate and maintain the software. The finance perspective helps identify income from software usage and expenditures on infrastructure facilities.

A ubiquitous language is a set of unambiguous vocabularies shared by all stakeholders in the domain.[25] In order to successfully develop a software, it is necessary to define it as early as possible. The customer perspective can help build a shared vocabulary. Key activities in BMC enable the elicitation of high-level requirements for the software that will be developed. Thus, rough boundaries about domains can be learned from key activities.

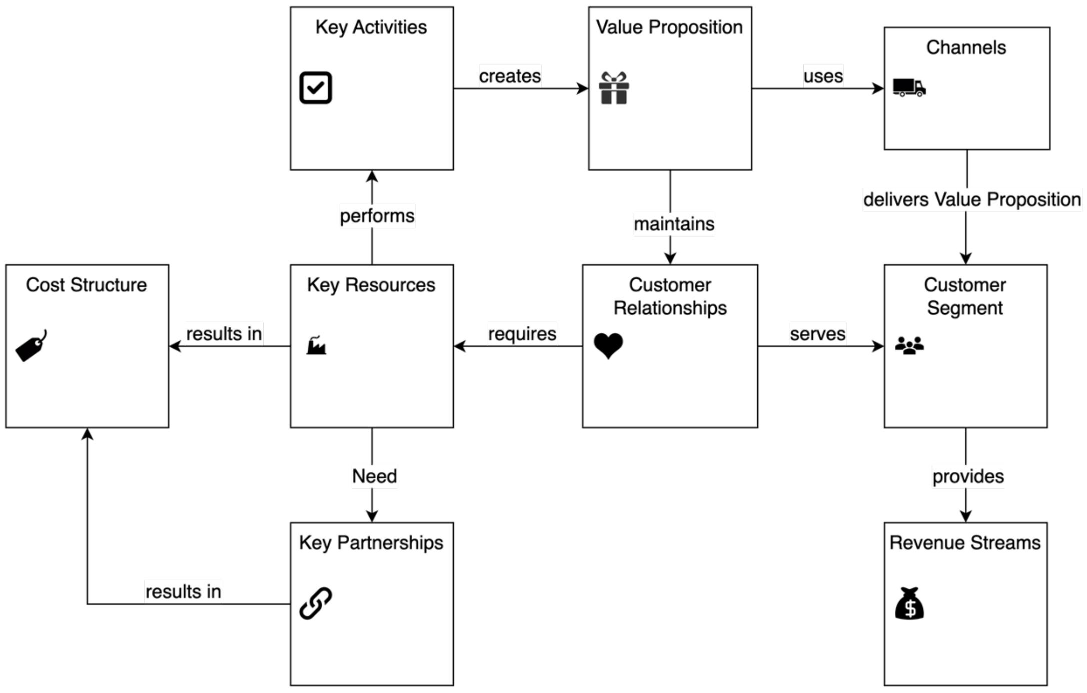


Figure 3.1. Meta-model for BMC elements and relations

An example BMC diagram is depicted in Figure 3.2. In this example, software to manage and automate Internship Application in educational institutes is modeled using BMC:

- Value proposition: Software will maintain and automate internship applications in education institutes.
- Customer Segments: The software will be used by educational institutes.

- Customer Relationships: A support mechanism for customers will be held to resolve issues when using the software.
- Channels: The software will be web based and be served SaS (software as a Service)
- Key Activities: The software has the following functionalities; internship announcement management, student internship application management, and company internship opportunity management.
- Key Resources: Software developers will maintain the software, and support persons will support customers.
- Key Partnerships: As the software will be run on a cloud server, a cloud service providers will be key partner.
- Cost Structures: Software developers, Support persons, and Cloud service providers will be expenses.
- Revenue Streams: Subscription of Educational institutes to the software will be income.

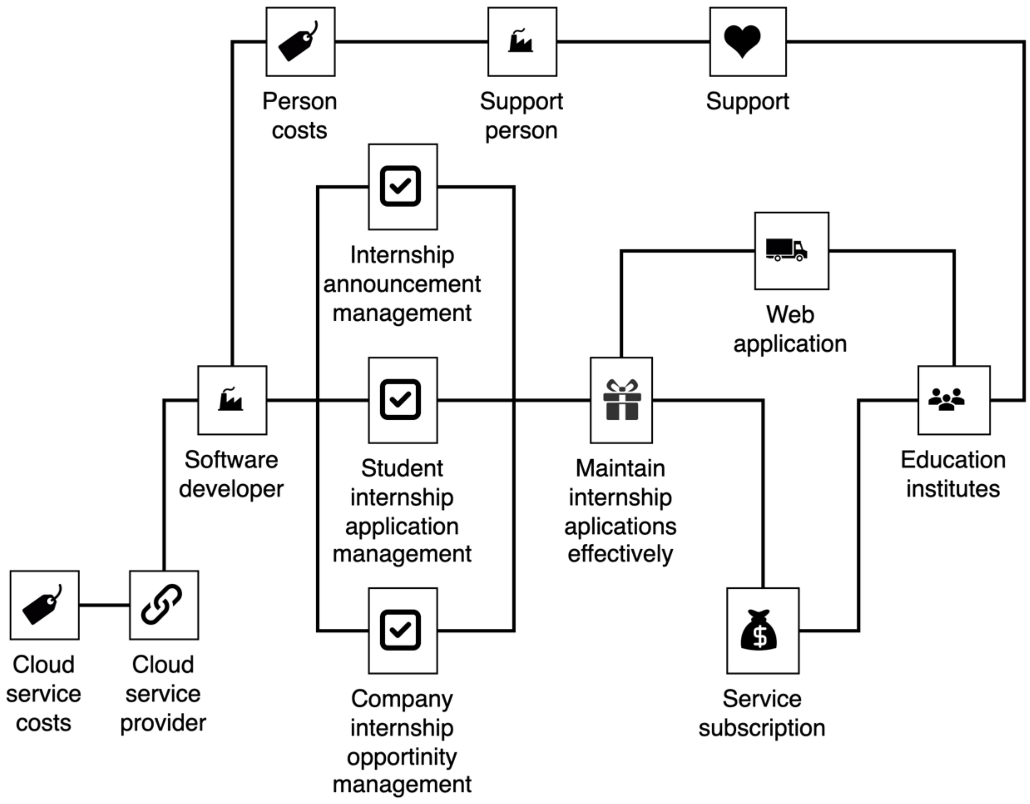

Person costs
Support person
Support
Internship announcement management
Web application
Software developer
Student internship application management
Maintain internship aplications effectively
Education institutes
Cloud service costs
Cloud service provider
Company internship opportinity management
Service subscription

Figure 3.2. BMC diagram

3.1.2. Event-Driven Process Chain

An Event-Driven Process Chain (EPC) is a flow diagram used to model business processes.[26] The core elements of the EPC are Event, Process and Logical Connectors (XOR, AND, and OR) (Figure 3.3).

Event is a state that controls or influences the progression of the process. Event triggers the process, or is produced by the process. Process is the task or activity performed to achieve an objective. Logical Connectors are used to control the flow. The Flow is split or joined using the connectors. XOR considers only one path, AND considers all paths, and OR considers at least one path.

EPC allows you to model process flows from high level to low level. In EPC, a process can have a sub-process, so a process can be modeled in more detail. This increases the clarity of models.

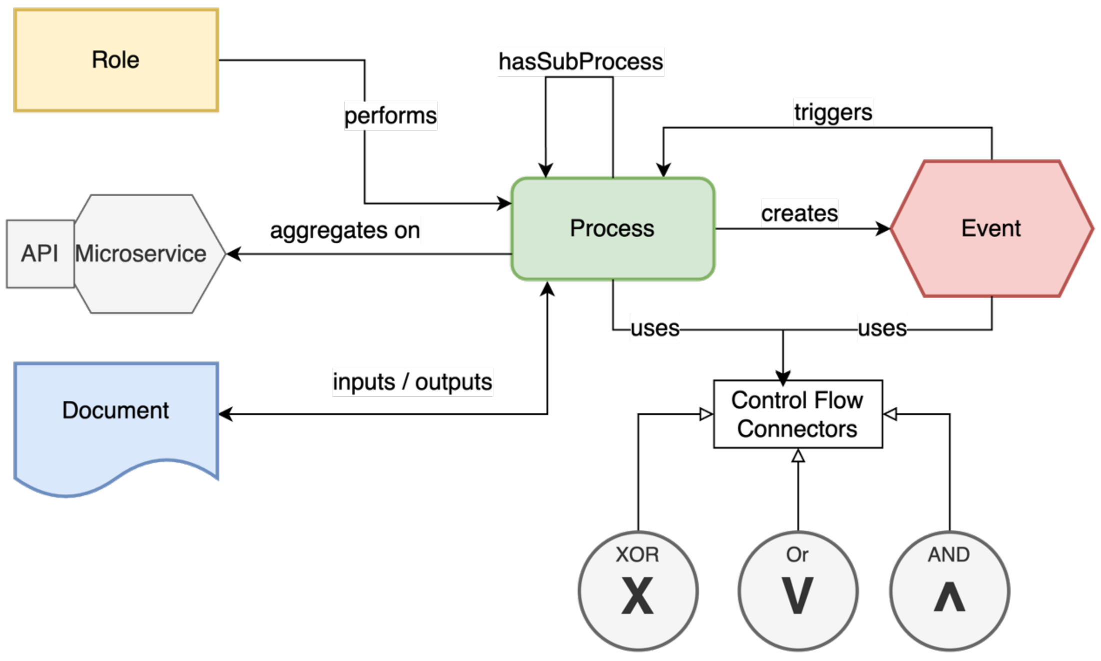


Figure 3.3. Meta-model of EPC elements and relations

A business process is modeled as a sequence of activities. It begins with an event that triggers the process. When a process completes its task, it produces an event that triggers another process. An example EPC diagram is depicted in Figure 3.4. In this example, a customer uses an e-shopping application to place an order. He/she places an order. If the requested product is in stock, then it is prepared for shipping, and inventory is updated by reducing the number of products in stock. If the requested product is out of stock, it is reordered, and the customer is notified of the late shipment.

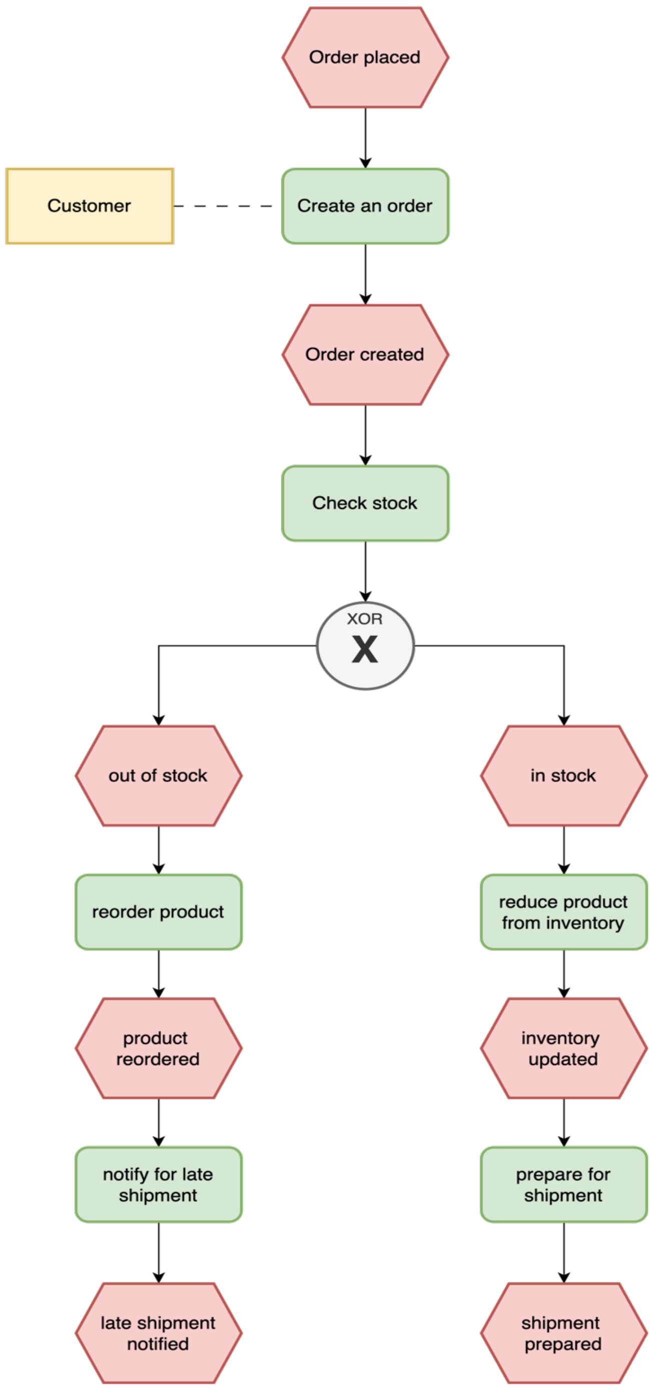


Figure 3.4. EPC diagram

3.1.3. Microservices Event Flow Diagram

The Microservices Event Flow Diagram (MEFD) enables the modeling of microservices and their interactions, including event streams and auxiliary components such as Event Brokers, API Gateways, Service Registries, etc. (Figure 3.5). It provides a high-level overview of how various components communicate and collaborate in a micro-service-based system.

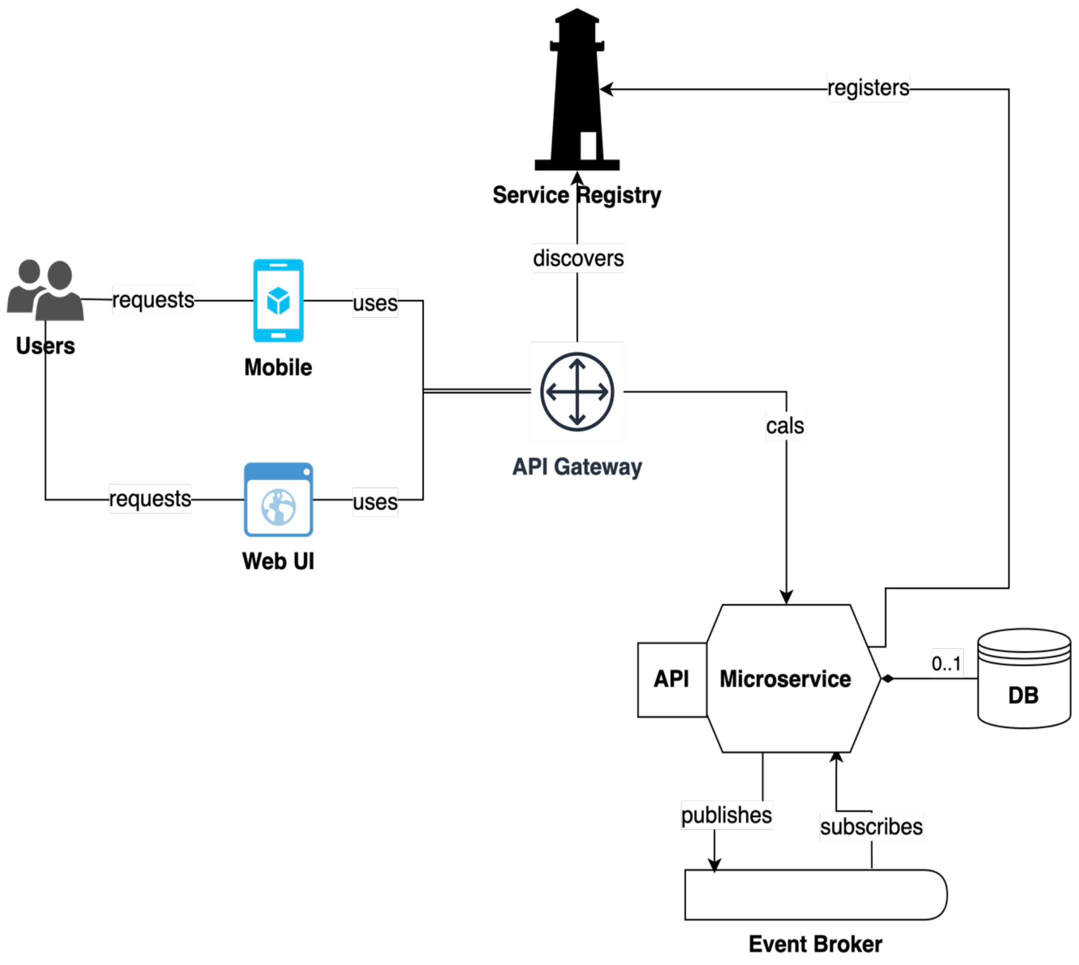


Figure 3.5. Meta-model for MEFD elements and relations

An example Microservice Event Flow diagram is depicted in Figure 3.6. In this example, the Internship Management software is decomposed into five microservices and these microservices communicate asynchronously with each other using an event broker that handles event flows.

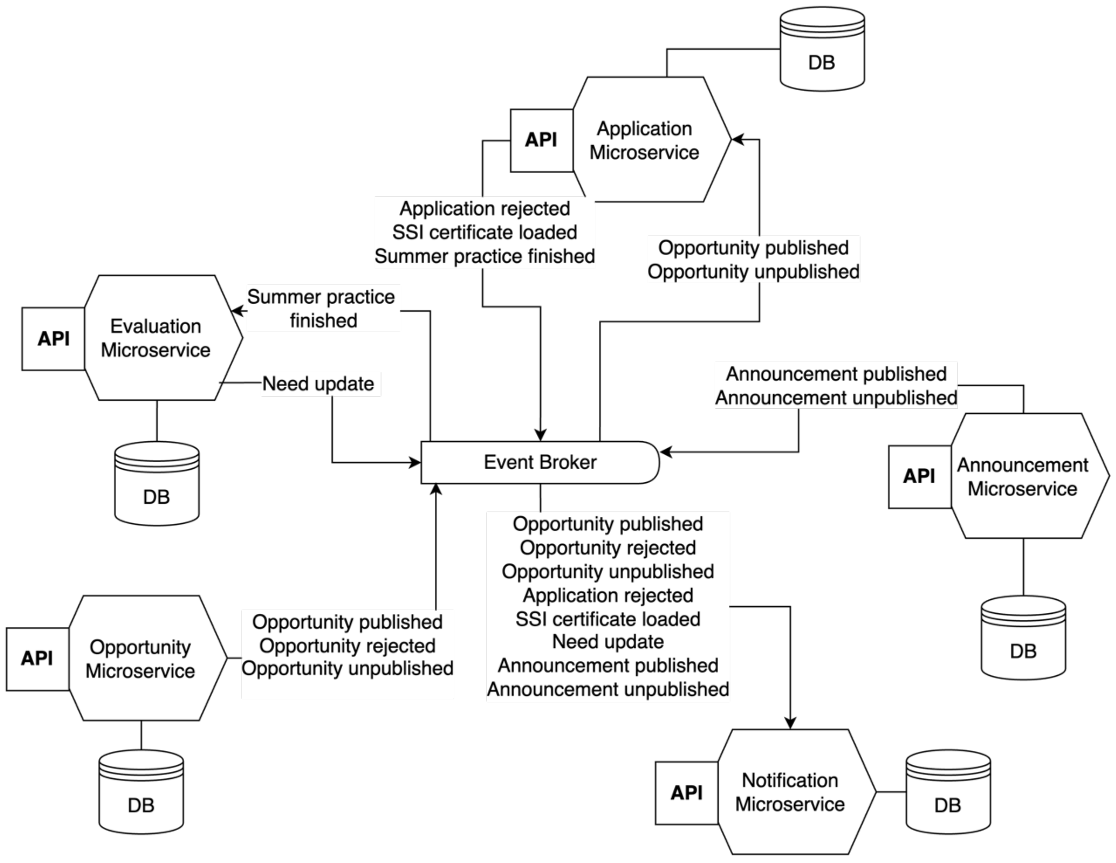


Figure 3.6. Microservice Event Flow diagram

3.1.4. Integration of Diagrams

MicroArc allows for the transition between models. In BMC model, the flow of key activities (business processes) was modeled using the EPC model. Additionally, in the MEFD, the flow of a microservice's activities (business processes) is modeled using the EPC model. Therefore, integration between models can be established, and these changes can be reflected in the connected models. Interactions between models are depicted in Figure 3.7.

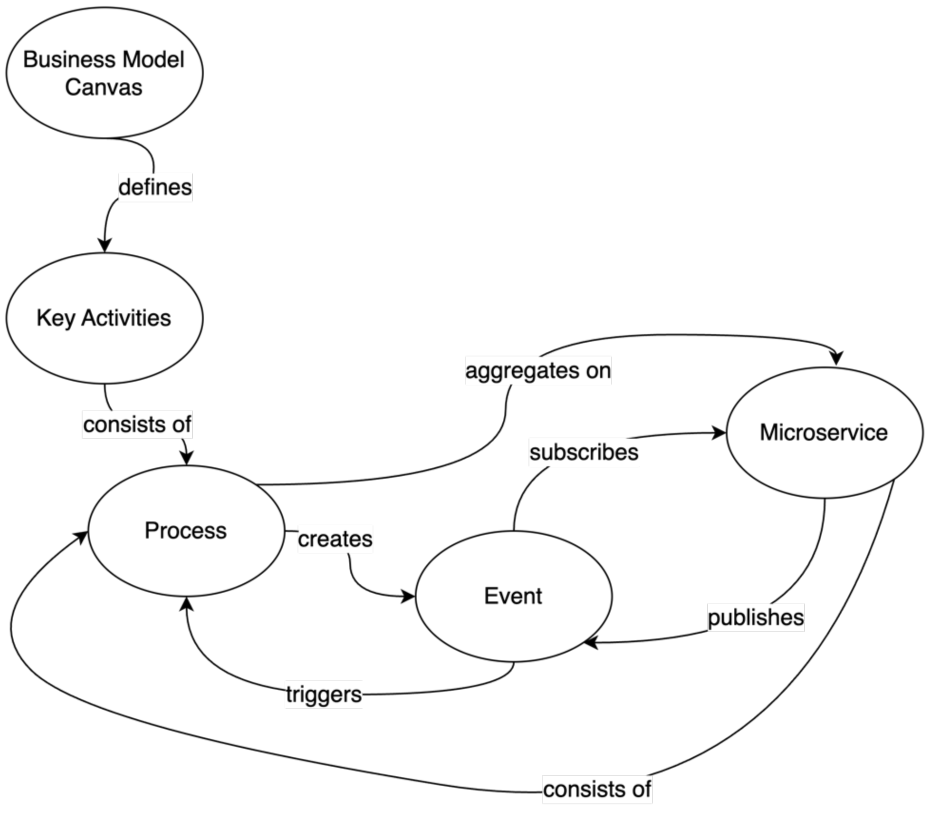


Figure 3.7. Interactions between models

3.1.5. Implementation Steps of MicroArc Method

MicroArc method consists of the phases of modeling a business idea, business processes, and event flows between microservices (Figure 3.8).

Business idea modeling is based on defining the business idea from different perspectives using BMC. Key Activities in BMC provide information about the business domain and define high-level system functionalities. Ubiquitous language, which is important for establishing a common understanding between stakeholders, can be identified from key activities.

Business process modeling is based on defining the flow of the activities of a system's functionalities using EPC. EPC enables the identification of events and activities (process) of the functionalities. In EPC, events enable the definition of boundaries of microservices based on the high cohesiveness of activities in a process flow. EPC can model the flow of activities at different levels of detail. An activity (process) in EPC can have a sub-model (sub-process) to model its internal flow in more detail.

Event flow modeling is based on showing event flows between microservices using MEFD. MEFD shows published events that are published from microservices and subscribed events that are subscribed to by microservices.

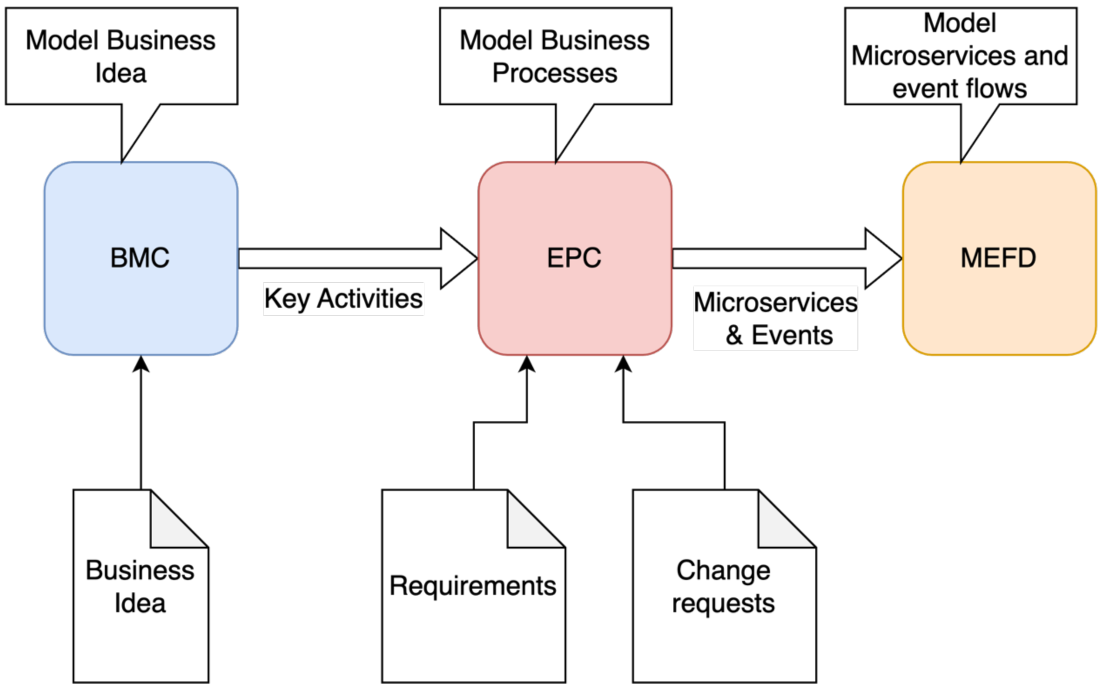


Figure 3.8. MicroArc method implementation phases

The Analysis and Design of microservices based systems using MicroArc Method involves the following steps:

1. **Build BMC of software system:** The software system is initially conceptualized as a business idea, and then a Business Model Canvas is created. The BMC model helps identify the key activities related to the system.
2. **Identify High-level Functionalities of software system:** The key activities identified in the BMC model are considered as high-level system functionalities.
3. **Establish EPC for the functionalities of software system:** An event-driven process chain model is employed to conduct a detailed analysis of key activities.

The EPC model defines the flow of each key activity by illustrating events, activities (processes), and control points. This modeling technique offers a visual representation of the system processes and their relationships.

4. **Aggregate activities based on high cohesion:** Activities identified in the EPC model are aggregated based on their high cohesion. One of the main characteristics of a microservices cohesion. Microservices encourages high cohesive architecture. Cohesion refers to how well the individual elements within an application work together. This aggregation step aims to group the related activities, enhance modularity, and reduce dependency. The functionality of a microservice consists of processes working in cohesion.
5. **Identify Microservices and related events:** Once activities are aggregated, candidate microservices are identified based on these aggregated activities. Selecting Microservices based on aggregated activities helps to define the boundaries and responsibilities of each service.
6. **Establish MEFD of microservices:** The final step involves using the Microservices Event Flow Diagram to model the interaction between microservices and illustrate event streams. MEFD provides a visualization of how microservices would communicate with each other by exchanging events.

Although the MicroArc model appears to have sequential steps, it can handle any requirement change. When a requirement change occurs to update an existing functionality or to add a new functionality, the flow of activities of this functionality is re-modeled using EPC. The newly created EPC model is then reviewed to identify new events that can redefine the boundaries of microservices. Boundary changes can initiate a new microservice or update its behavior.

3.2. MicroArc Tool

A web-based tool to support MicroArc model has been developed. The MicroArc tool enables analysts to use MicroArc method notations to perform event-driven analysis and design for Microservice based architectures. The MicroArc tool is implemented using

Eclipse Sirius platform. Eclipse Sirius allows creating a graphical modeling workbench using Eclipse Modeling Framework (EMF). EMF enables the creation of custom models that include rules and validations within the created model.

This section presents the high-level functional requirements of the MicroArc tool and its architecture.

3.2.1. High Level Requirements

The high-level functional requirements of the MicroArc tool that supports the MicroArc notation are depicted as a use case diagram in Figure 3.9.

Main functionalities of MicroArc tool are classified as Project related, and Model related. These functionalities and their definitions are given below:

- *Create Project:* The analyst creates a project to which models are assigned. The project can be a new project that will be implemented or a new version of a project. In the MicroArc tool, models are related to a project.
- *List Project:* Created projects are listed to open them to create models. Analysts have the ability to perform various operations such as deletion, renaming, and download on the listed projects.
- *Delete Project:* The analyst can delete a project from the listed ones. When a project is deleted, all models belonging to it are also deleted.
- *Download Project:* All projects in MicroArc are stored in a database. Analyst can download a project, if he/she wants to store it in his/her storage. When a project is downloaded, all models belonging to it also get downloaded.
- *Upload Project:* Analyst can upload a project from his/her storage to MicroArc tool. The uploaded project is stored in a database. When a project is uploaded, all models belonging to it are also uploaded and stored in the database.
- *Rename Project:* Analyst can rename a project with a desired name.
- *Create BMC Model:* BMC model that is belongs to a project is created. Analyst uses BMC model elements for model creation and connects related ones with connectors.

- *Create EPC Model:* EPC model that is belongs to a project is created. Analyst uses EPC model elements for model creation and connects related ones with connectors.

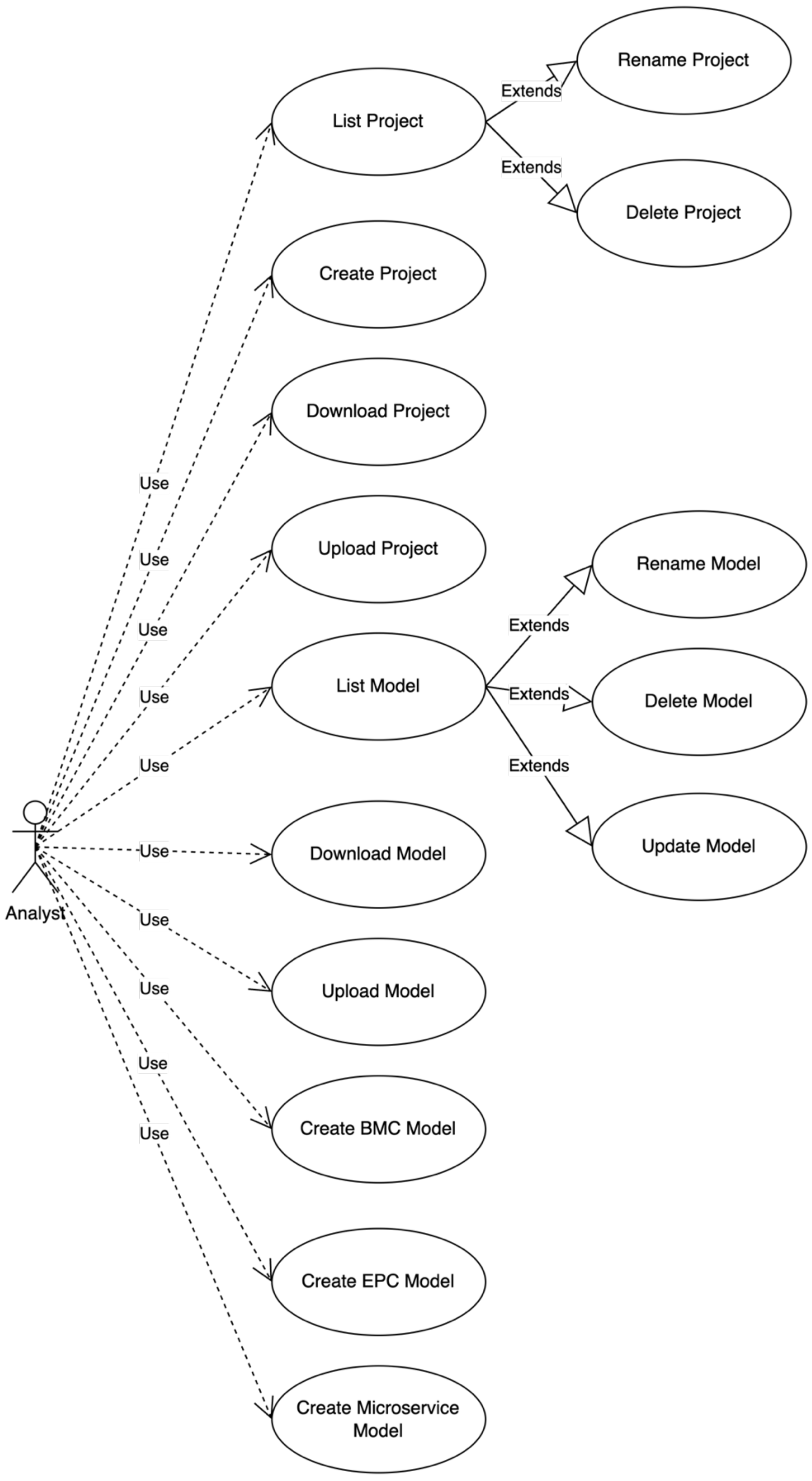


Figure 3.9. MicroArc tool Use-cases

- *Create Microservice Model:* Microservice model that belongs to a project is created. Analyst uses Microservice model elements for model creation and connects related ones with connectors.
- *List Model:* The created models are listed, allowing analyst to open them and create new ones. Analyst can make some operations (delete, rename, update, download) on listed models.
- *Delete Model:* Analyst can delete a model from the listed ones.
- *Rename Model:* Analyst can rename a model with a desired name.
- *Update Model:* Analyst can update a model to rearrange its model element.
- *Download Model:* All models in MicroArc are stored in a database. Analyst can download a model if he/she wants to store it in his/her storage.
- *Upload Model:* Analyst can upload a model from his/her storage to MicroArc tool. Uploaded model is stored in a database. When a model is uploaded, it belongs to a project.

3.2.2. Architecture of the Tool

The MicroArc tool is developed as a web application that uses a database to store models. The user interface of MicroArc tool has three areas: Model Explorer, Model View, and Details View (Figure 3.10).

- *Model Explorer* allows analyst to manage models of a project. Analyst can see a list of models with model's elements and create, delete, update these models.
- *Model View* allows analyst to draw model by using Contextual Palette which includes the model's elements. Analysts can also create various visualizations on models, including zooming in and out, full screen mode, importing SVG files, and arranging all elements.
- *Details View* enables analyst to see properties of the model, and the model's elements. It also allows analyst to update properties.

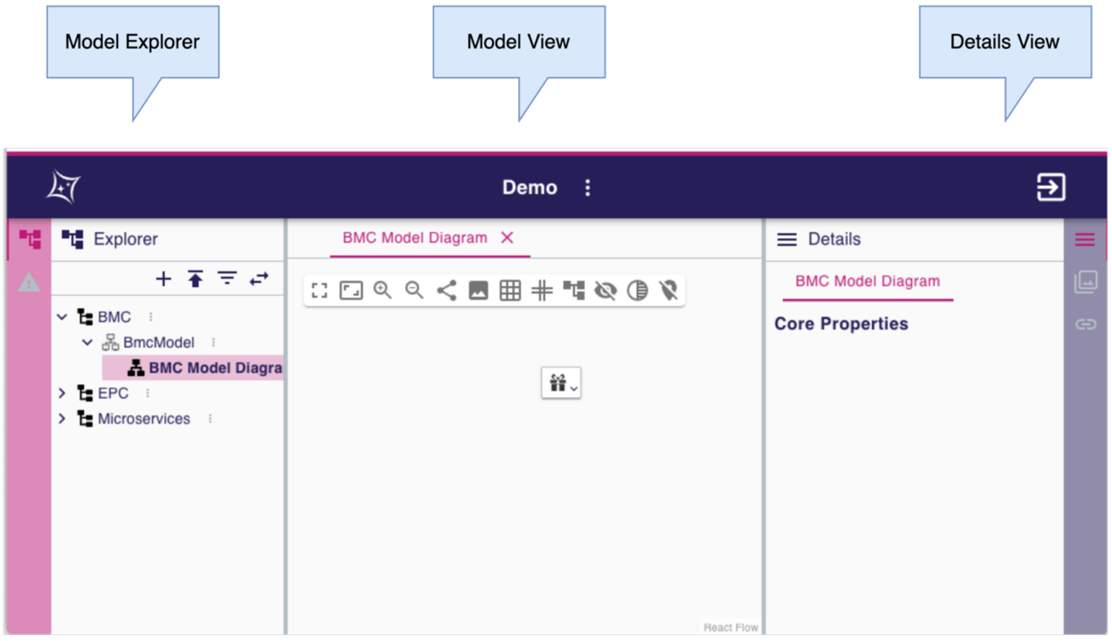


Figure 3.10. MicroArc tool User-interface

CHAPTER 4

CASE STUDIES

In this chapter, Case Studies (Exploratory and Explanatory) with their design and results are presented respectively. The overall discussion of these studies is given in the last section.

4.1. Exploratory Study 1- Comparison of OO and EO

This exploratory study was performed to determine if the Object Oriented or Event Oriented analysis approaches provide a good strategy for developing Microservices based Systems (MSbS). In this study, two approaches were compared with respect to the key MSbS characteristics, and the advantages and difficulties of each approach were discussed.[7]

4.1.1. Research Question in Exploratory Study 1

Following research question, related proposition, and validation method were defined for exploratory study 1:

- *Research question*: *How successful are Object Oriented Analysis and Design (OOAD) and Event Oriented Analysis and Design (EOAD) in meeting the key characteristics required by a Microservices based Architecture?*
- *Proposition:* The goal is to observe how OOAD or EOOD fulfill key MSbS characteristics and what the advantages and difficulties of each approach are. This

proposition would be validated by implementing a case study with two approaches.

- *Validation method for proposition:* Key MSbS characteristics will be evaluated whether they are met sufficiently by OOAD or EOAD.

4.1.2. Activity Planning in Exploratory Study 1

The following activities were planned to performed within case study:

1. *Case Selection:* Four case selection criteria will be established in order to effectively reach the research goal and satisfy the case study validity concerns. The first criterion is that the selected case should include processes that can be automated. The second criterion is that there should be specific events that will trigger each process. The third criterion is that the case should be large enough to implement at least two microservices; nonetheless it should be small enough to be implemented as whole in an adequate time. The last criterion is that the selected case should be sufficiently detailed described so that it can be implemented without struggling with problem domain details.
2. *Performing Event Oriented Analysis and Design (EOAD):* Selected case will be performed using the EOAD approach.
3. *Performing Object Oriented Analysis and Design (OOAD):* The selected case will be performed using the OOAD approach. Two teams will implement the OOAD approach to prevent potential bias in the implementation both approaches with the same team.
4. *Analyzing the result:* Evaluation will be made based on implementations of both EOAD and OOAD approaches on the case, which will then be compared with prominent characteristics of MSbA.

4.1.3. Mitigation of Threats to Validity in Exploratory Study 1

The exploratory case study was performed with IzTech third-year students who participated in OOAD and with a team who participated in EOAD, which is defined as a graduate course in IzTech. Although the third-year student team was selected with some experience with microservices, it can be thought they are relatively inexperienced. In order to prevent potential bias of team implemented both approaches, the third-year student team was involved as a second group in OOAD.

4.1.4. Exploratory Study 1 Design and Execution

4.1.4.1 Case Selection in Exploratory Study 1

“The graduate student application and evaluation process” at IzTech was selected as a case that satisfies all four criteria mentioned in the case activity planning section. First, it is suitable for automation in a distributed execution manner by consisting of fair enough small and isolated subprocesses. Second, all subprocesses can be triggered from predefined events and executed in cohesion. Third, it is large enough to be implemented in a semester. Last It is modeled by using the well-defined instructions on IzTech’s web page. IzTech’s application to graduate programs process activities are given in Table 4.1.

Table 4.1. IzTech graduate student application activities

Activity	**Description**
Application	Criteria to apply to master or doctorate programs for applicants are defined, and it defines how and when to apply.
Verification	Applicants are verified based on application criteria.
Evaluation	Applicants are evaluated by performing an interview or/and exam.
Notification	Applicants who are successful are notified to enroll.

4.1.4.2 Performing Event Oriented Analysis and Design (EOAD)

The selected case was performed by using EOAD methodology. The selected process is modelled (AS-IS) using eEPC (extended Event-Driven Process Chain) notation. eEPC mainly consists of events, functions (processes), connectors (xor, and, or), and other views (such as roles, product/service, Data Input/Output, etc.). Event triggers a function (process) and the function produces an event. Process flow is modelled as event driven process chain. eEPC allows modeling a process flow in any detail (from high level to low level). Functions in a process flow can be modeled more detailed as sub-processes. The eEPC model of the AS-IS process of the selected case is depicted in Figure 4.1. The selected process starts with an announcement of application to the graduate programs, after applicants apply the programs, verification of the applicants starts. Verified applicants are interviewed and assessed. Finaly, the results are announced.

In order to identify the bounded context of the microservices from model, the TO-BE model of the process is modeled to become automated in Figure 4.2. In EOAD, events draw the boundaries of bounded context. A microservice consists of related (high cohesion) processes (activities), and, within this bounded context, processes perform their functionalities assigned to them within collaboration. Application, Verification, Evaluation and Notification microservices are identified from TO-BE model.

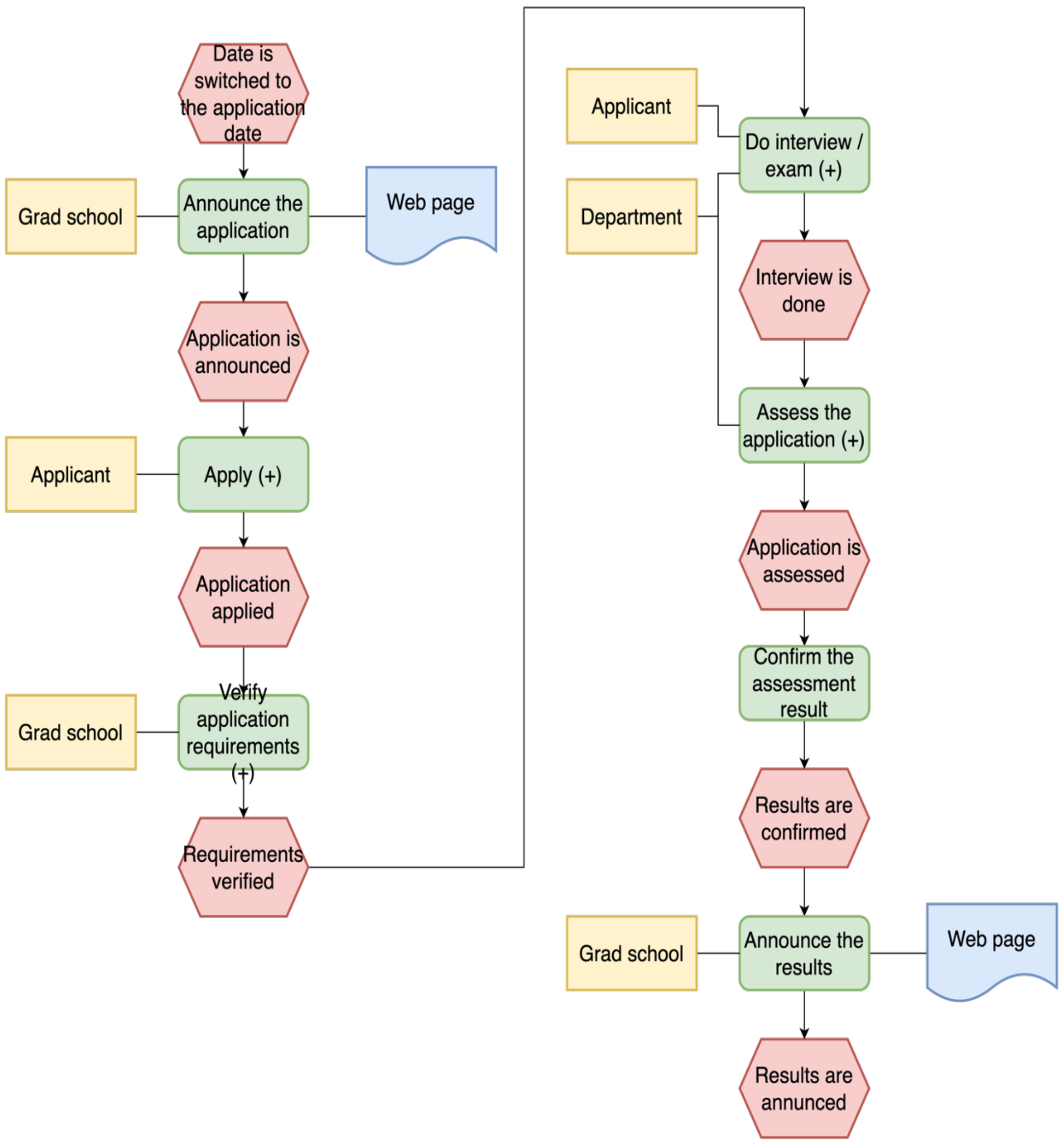


Figure 4.1. EPC for graduate student application AS-IS

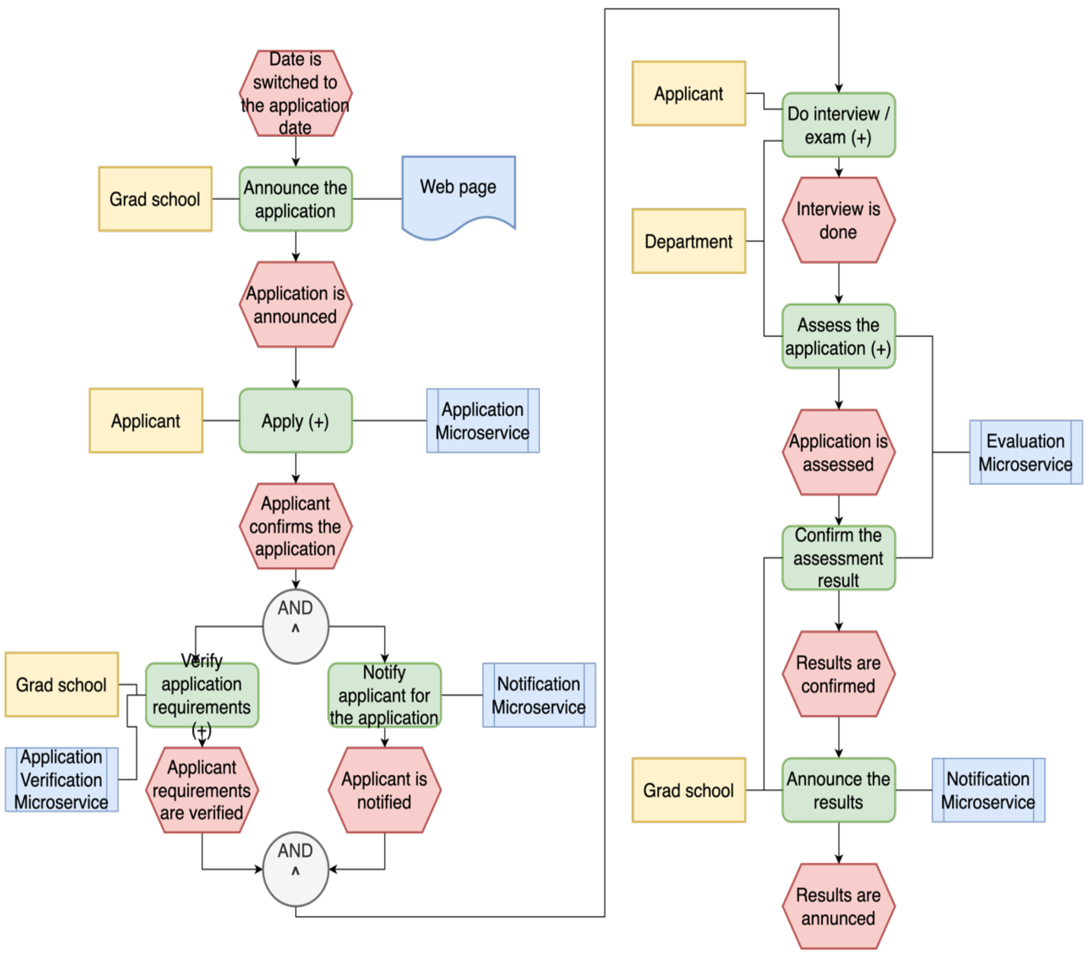


Figure 4.2. EPC for graduate student application TO-BE

4.1.4.3 Performing Object Oriented Analysis and Design (OOAD)

The selected case was performed by using OOAD methodology. First the use

The selected case was performed by using OOAD methodology. First the use cases were identified (Figure 4.3). Then the classes were identified in a class diagram. The class diagram provides a detailed representation of the main elements and the relations between them. Finaly the system behaviors were shown in activity diagram (Figure 4.4). Performing EOAD activities could have given some insight about Microservices and it could be bias for finding Microservices from OOAD approach. A different team from IzTech third-year students independently performed OOAD approach for same case.

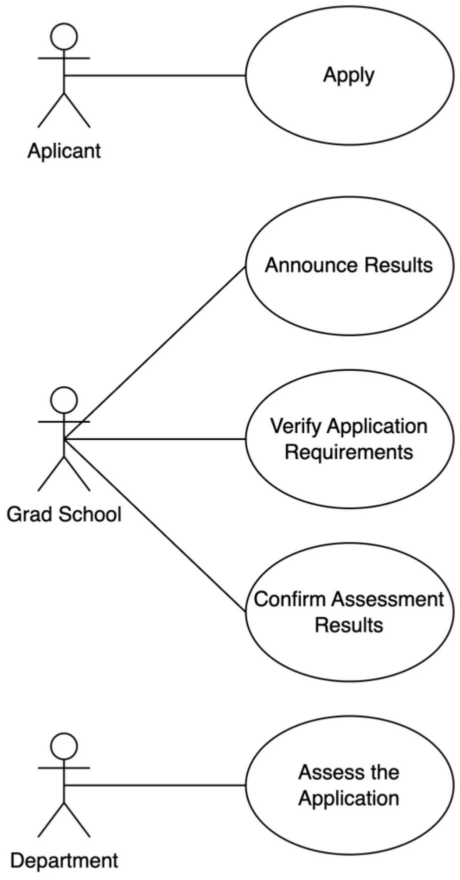


Figure 4.3. Graduate student application Use-cases

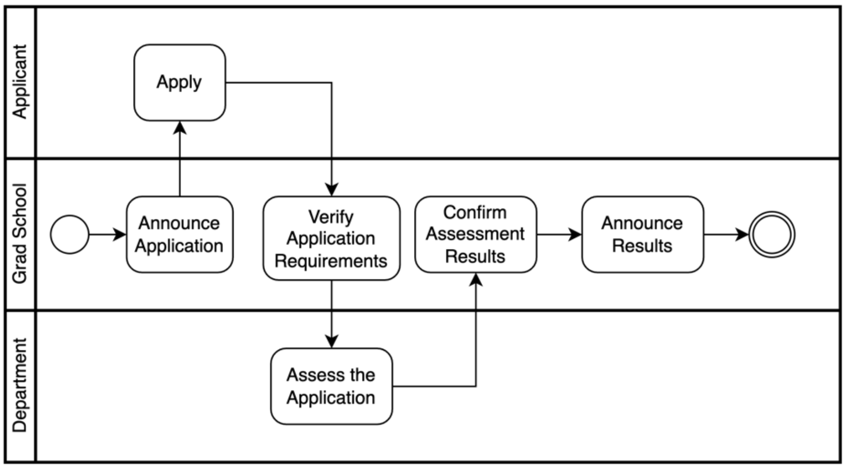


Figure 4.4. Graduate student application Activity diagram

4.1.5. Analysis and Results of the Exploratory Study 1

An exploratory case study was conducted and analyzed for exploratory purposes. In EOAD approach, microservices and related events can be found from EPC model; however, OOAD approach could not help to find microservices. As a result, OOAD was unable to provide a strategy for developing an MSbA for the given case.

The comparison and evaluation of two approaches (EOAD and OOAD) based on prominent characteristics of MSbS are discussed:

- *Loose coupling:* Object-Oriented Analysis and Design provides loose coupling at the class level but does not separate business capabilities. In EOAD, separation is based on business capabilities.
- *Cohesion:* Object Oriented Analysis and Design focuses on the cohesion between classes. Class-based decomposition does not help in keeping all the functionally related processes together. In EOAD, separation on business capabilities provides high cohesion for microservices.
- *Isolation:* It is not possible for Object Oriented Analysis and Design methods to create a completely isolated class structure based on class decomposition.
- *Asynchronous communication:* In Object Oriented Analysis and Design, communication is among the class methods that do not provide a strategy for asynchronous communication. In EOAD, using events for message transfer between microservices provides asynchronous communication
- *Single responsibility:* In Object Oriented Analysis and Design, the system is separated by classes. However, the sole responsibility of the class may not correspond to the microservice. In EOAD, separation on business capability defines the boundaries of responsibilities.
- *Fault tolerance:* In Object Oriented Analysis and Design a system is less tolerant to system failures as it causes an inter-process dependency between classes, shared methods and data attributes used by multiple sub-processes.

4.2. Exploratory Study 2- Establishing Microarc Method and Tool

This exploratory study was performed to establish an event driven analysis and design method, called MicroArc, for Microservices based systems. In this study tool requirements were identified to support MicroArc method.

4.2.1. Research Questions in Exploratory Study 2

Following research questions, related propositions, and validation methods were defined for exploratory study 2:

- *Research Question 1: How to discover microservices and related events from event-driven process chain (EPC) model?*
- *Proposition 1:* The goal is to define microservices and related events from EPC model. This proposition would be validated by implementing a case study.
- *Validation method for proposition 1*: Identified microservices will be evaluated whether they are high cohesive.
- *Research Question 2: How to discover business domain and their sub domains?*
- *Proposition 2:* The goal is to define sub domains of a software system. This proposition would be validated by implementing a case study.
- *Validation method for proposition 2*: Software system will be modelled using BMC and evaluated whether Key activities in BMC indicate sub domains.
- *Research Question 3*: *How to show event flows between microservices?*
- *Proposition 3*: The goal is to show event flows between microservices. This proposition would be validated by implementing a case study.
- *Validation method for proposition 3*: An event flow model will be defined with model elements and event flows between microservices will be modelled using this diagram.

4.2.2. Activity Planning in Exploratory Study 2

The following activities were planned to performed within case study 2:

1. *Case Selection*: Following case selection criteria will be established in order to effectively reach the research goals and satisfy case study validity concerns. (1) Case should have activities that can be automated. (2) Case should be clearly described so that it can be implemented without requiring a thorough understanding of the problem domain.
2. *Performing MicroArc method*: Selected case will be performed using MicroArc method, and supporting tool requirements will be identified.
3. *Analyzing the result:* Evaluation will be made based on implementation of MicroArc method on the case, then results will be analyzed whether the goals of research question have been met.

4.2.3. Mitigation of Threats to Validity in Exploratory Study 2

The exploratory case study was performed by researcher. He was experienced with Business Model Canvas (BMC), modelled many business processes using EPC, and attended to a graduate course in IzTech.

4.2.4. Exploratory Study 2 Design and Execution

4.2.4.1 Case Selection in Exploratory Study 2

“AI Estimator: AI Based Size and Effort Estimation Tool” definition was selected as a case that satisfies all criteria mentioned in the case activity planning section. (1) It has activities that suitable for automation. (2) It will be caried out in a research project, and has clear definition.

4.2.4.2 Performing MicroArc Method

The selected case was performed by using MicroArc method. The AI Estimator is a tool used to measure the size of a project using artificial intelligence models trained previously based on completed software projects. Software Organizations use AI Estimator to predict functional sizes of their potential projects and estimate their efforts. The AI Estimator provides Software Size and Effort Estimation, AI Model Training, and Benchmarking functionalities to users.

Initially, it was modeled using the BMC model which is used to illustrate the business perspective in Figure 4.5. Subsequently, key activities were modeled to identify the flow of these activities by employing the EPC. Table 4.2 describes key activities and their descriptions. An EPC diagram of the AI Estimator Model Training key activity is shown in Figure 4.6. Then, candidate microservices were discovered by aggregating the processes in the flow based on high-cohesion. Table 4.3 lists candidate microservices and their triggered and produced events. Finally, events triggered and produced by the microservice were selected, as shown in Figure 4.7.

Table 4.2. AI Estimator Key Activities

Key Activities	**Description**
Software size and Effort estimation	Software size and effort estimation enable companies to estimate the size and effort of a project based on requirements. Size and Effort estimation can be performed using either custom-trained or pre-trained AI models.
AI Model Training	Size and Effort estimations are made using AI Models. AI models are trained using previously labeled project requirements and classified based on project types, software development lifecycles, and programming languages.
Benchmarking	Benchmarking allows companies to compare their effort per size values with other companies based on project types, development languages, life cycles, and so on.

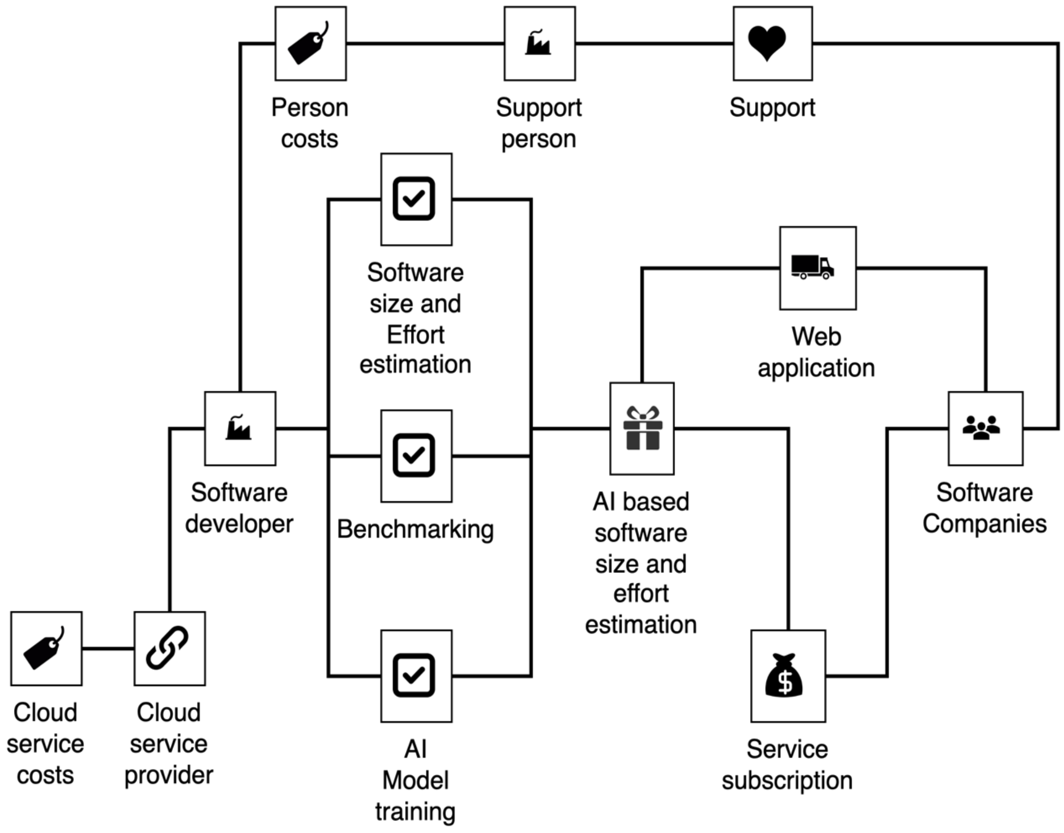


Figure 4.5. BMC model of AI Estimator

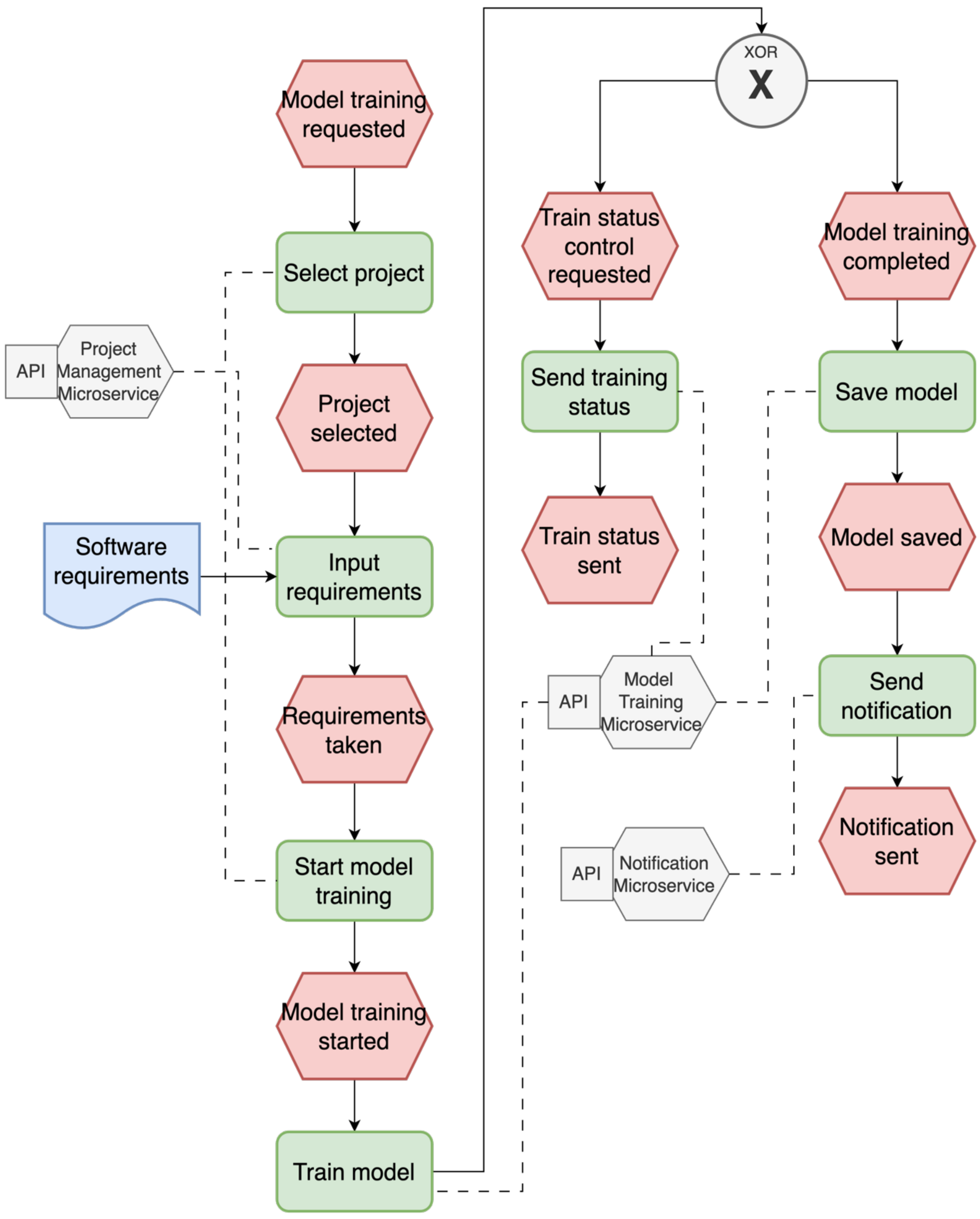


Figure 4.6. EPC of AI Estimator Model Training

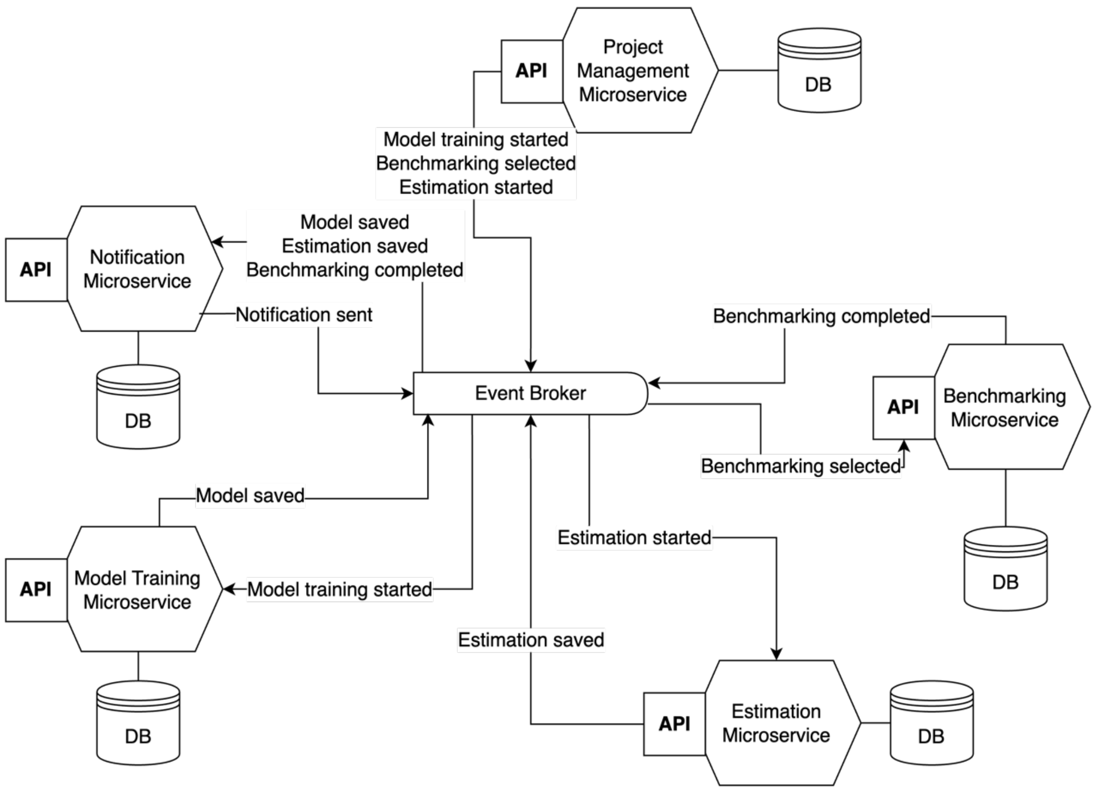


Figure 4.7. MEFD of AI Estimator

Table 4.3. AI Estimator microservices definitions

Microservices	Cohesion	Triggered Event	Produced Event
Project Management	Project related activities	-	Model training started, Benchmarking selected, Estimation started
Model Training	Model training activities	Model training started	Model saved
Estimation	Estimation activities	Estimation started	Estimation saved
Benchmarking	Benchmarking activities	Benchmarking selected	Benchmarking completed
Notification	Notification activities	Model saved Estimation saved Benchmarking completed	Notification sent

4.2.5. Analysis and Results of the Exploratory Study 2

The case study was performed and analyzed for exploratory purposes.
The use of BMC provides the following benefits:

1. Key Activities give information about sub domains of AI Estimator
2. Key Activities give information about ubiquitous language that is used in sub domains. They describe high-level functionalities of a system, knowledge within these functionalities describe ubiquitous language

During the case study key activities were modeled by using EPC to extract the flow of activities. EPC is event-driven modeling approach, an activity flow is described as event-process chains. Event triggers a process and initiates it to perform its job, after the process completed its job, it produces a new event that can trigger other process. One of the main characteristics of a microservices cohesion. Microservices encourages high cohesive architecture. Cohesion refers to how well the individual elements within an application work together. It was seen that aggregating processes based on cohesiveness can indicate a microservice. The functionality of a microservice consists of processes working in cohesion.

Identified microservices and related events were modelled by using MEFD to show event flows between microservices. MEFD helps to model synchronous or asynchronous communications, published events that produced by microservices, and subscribed events that subscribed to by microservices.

Based on case study results, tool requirements were elicited. Supporting MicroArc tool should have following functionalities:

1. Tool should support BMC, EPC and MEFD modeling.
2. Tool should have a model view that provides a modeling environment
3. Models should be stored in permanent storage
4. Integration between models should be supported.
5. There should be a microservice icon in EPC, so aggregated processes can be linked to microservice icon.

4.3. Explanatory Case Study

This explanatory study was performed to validate the MicroArc method by comparing with another study that uses the same problem definition and was developed by İzTech undergraduate students using the Microservice Architecture.

4.3.1. Research Questions in Explanatory Study

Following research question, related proposition, and validation method were defined for explanatory study:

- *Research Question*: *What are the perceived benefits of MicroArc Method and tool when developing an event-driven microservices software?*
- *Proposition:* Using MicroArc method and tool would bring benefits such as discovering microservices and related events, reflecting changes between models.
- *Validation method for proposition*: Analysis and design of case will be made using MicroArc method and results will be compared with another study that was developed by undergraduate students to evaluate effectiveness of MicroArc method.

4.3.2. Activity Planning in Explanatory Study

The following activities were planned to performed within case study:

1. *Case Selection:* Three case selection criteria will be established in order to effectively reach the goals and satisfy case study validity concerns. First, case should have activities that can be automated. Second, case should be clearly described so that it can be implemented without requiring a thorough

understanding of the problem domain. Last, case should be implemented by others and its architecture should include event-driven microservice architecture.

2. *Performing case study:* Selected case will be performed using MicroArc method and compared with other study that implemented same case.

4.3.3. Mitigation of Threats to Validity in Explanatory Study

The exploratory case study was performed by the researcher and IzTech undergraduate students who implemented the same case. The researcher and students have basic knowledge about EPC.

4.3.4. Explanatory Study Design and Execution

4.3.4.1 Case Selection in Explanatory Study

"Summer Internship Management process" at IzTech was selected as a case that satisfies all three criteria mentioned in the case activity planning section. First, it is suitable for automation in a distributed execution manner by consisting of fair enough small and isolated subprocesses. Second, it is modeled by using the well-defined instructions on IzTech's web page. Last, case was implemented by IzTech undergraduate students as a homework.

Summer Internship Management process has following requirements:

- Students should be informed about the internship rules,
- The documents required for the internship should be organized and updated, and made them accessible to students,
- Internship opportunities from companies should be announced to students,
- Students' SGK declarations should be submitted,

- Students' internship documents should be collected,
- Students should be graded according to their competence in the internship

4.3.4.2 Performing MicroArc Method

The selected case was performed by using MicroArc method. MicroArc method consists of 6 steps. First, Summer Internship Management process was modelled using BMC. Second, Key Activities were identified as high-level functionalities (Figure 4.8). Third, the flow of each key activity was modelled by using EPC, and events, activities (processes), control points, that are related with a key activity, were elicited. Fourth, related activities within a key activity were aggregated based on their high-cohesion (Figure 4.9). Fifth, Aggregated activities were selected as candidate microservices. Last, MEFD was created to model event flows between microservices (Figure 4.10).

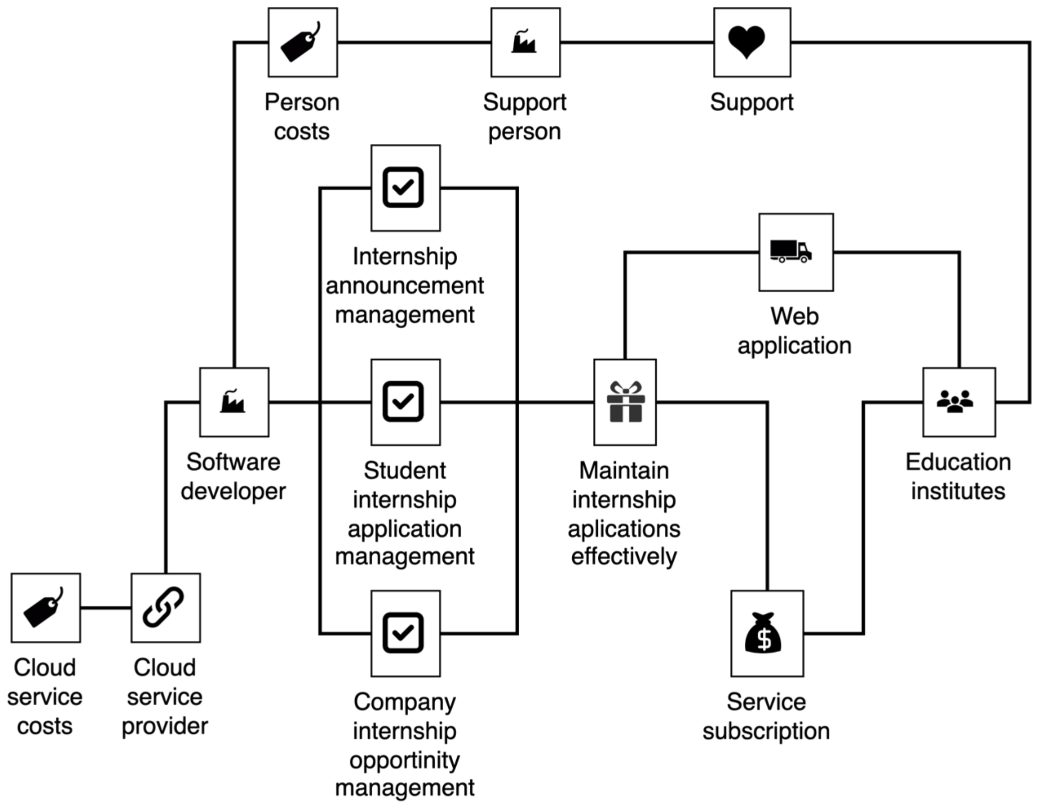

Person costs
Support person
Support
Internship announcement management
Web application
Software developer
Student internship application management
Maintain internship aplications effectively
Education institutes
Cloud service costs
Cloud service provider
Company internship opportinity management
Service subscription

Figure 4.8. BMC of Summer Internship Management

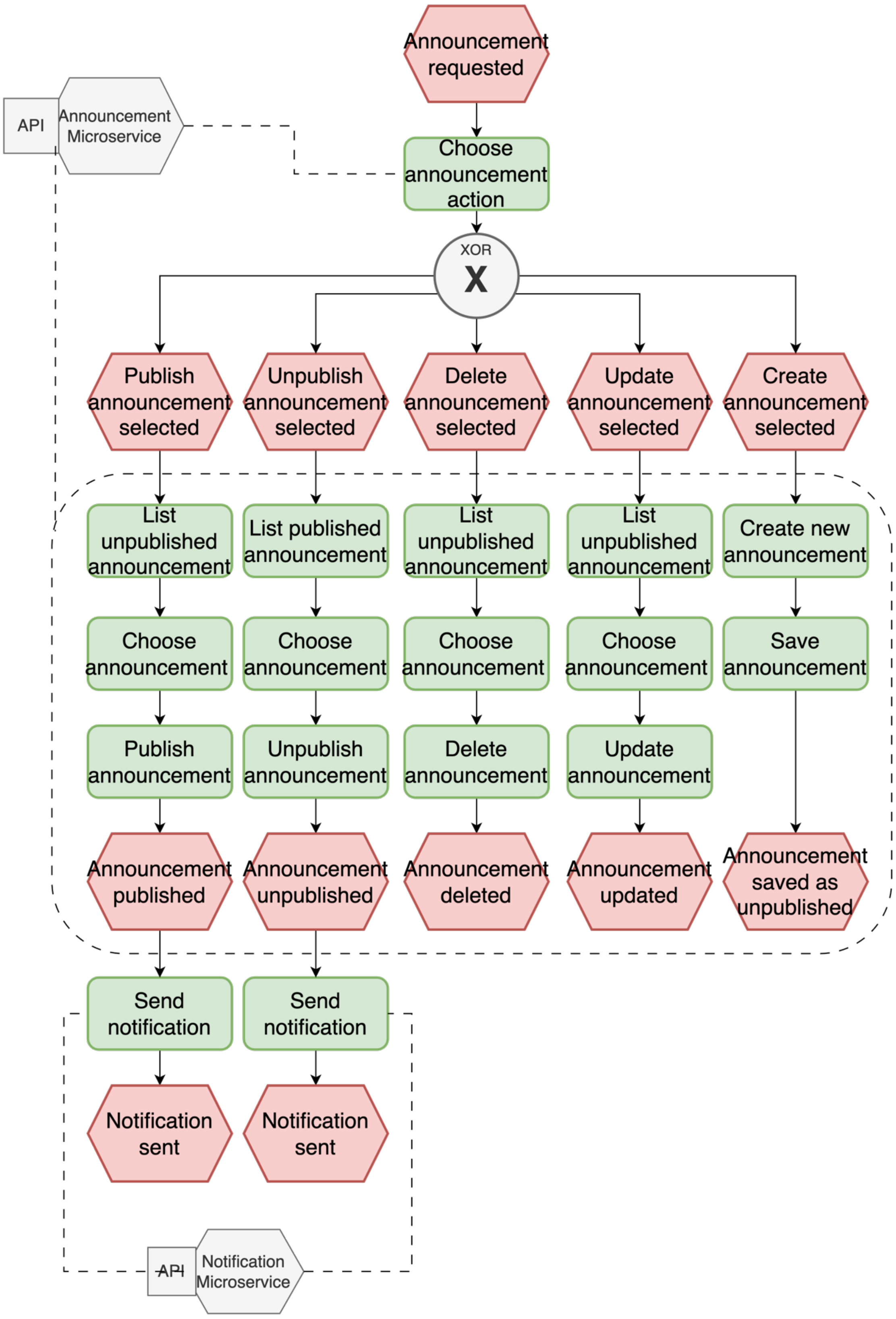


Figure 4.9. EPC of Internship Announcement Management

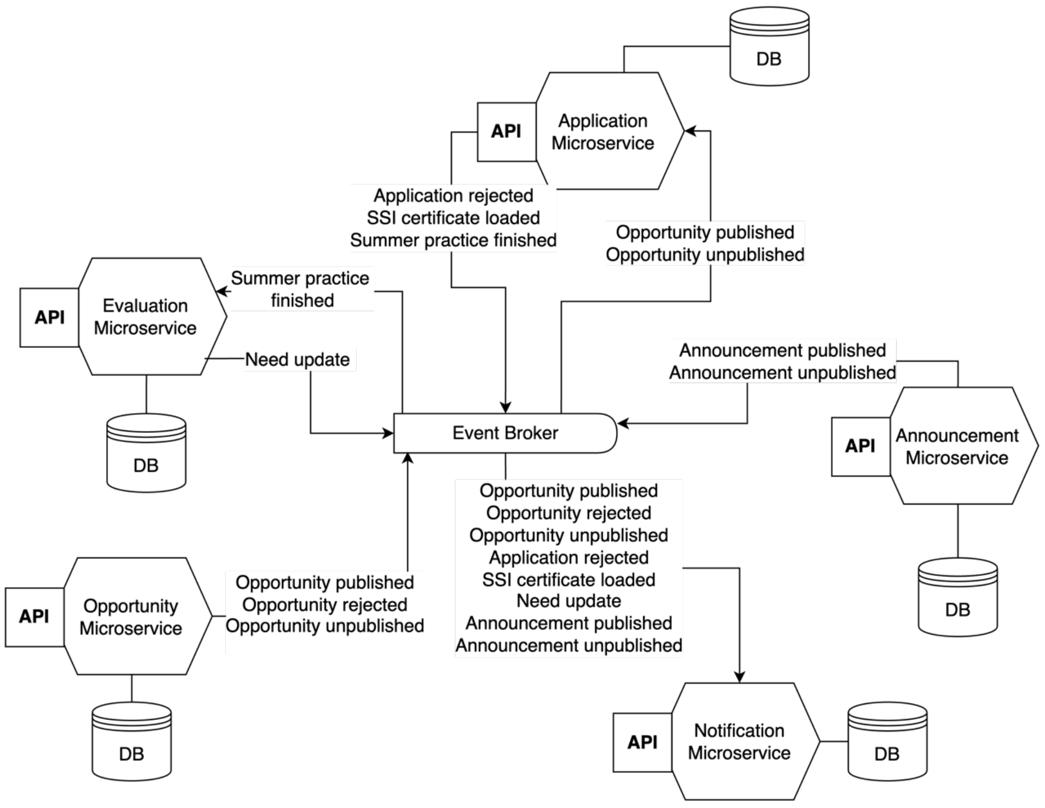


Figure 4.10. MEFD of Summer Internship Management

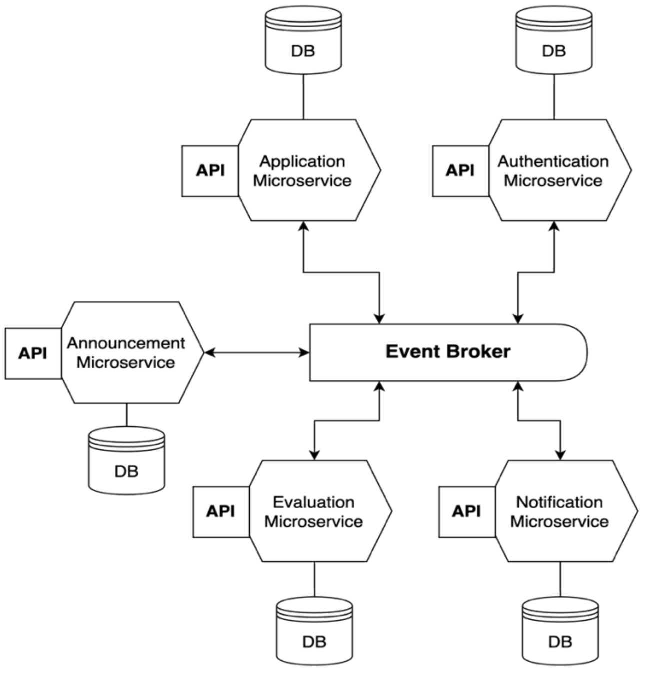


Figure 4.11. Undergrad Team Microservices for Summer Internship Management

4.3.5. Analysis and Results of the Explanatory Study

The explanatory case study was performed and analyzed for explanatory purposes. The comparison and evaluation of two studies based on effectiveness of MicroArc method.

Microarc and other study (undergrad team) have used EPC to model internship activities. Both studies have found similar microservices (Figure 4.10 and Figure 4.11). Undergrand team uses the same microservice for handling internship announcements and company opportunity requests; on the other hand, MicroArc suggests different microservices. The microservices and events for both study is depicted in Table 4.4 and Table 4.5.

Table 4.4. MicroArc's Microservices and related events of Summer Internship

MicroArc Microservices	MicroArc Events
Application Management: for handling application requests	Pub: Application rejected, SSI certificate loaded, Summer practice finished Sub: Opportunity published, Opportunity unpublished
Opportunity Management: for handling company opportunity requests	Pub: Opportunity published, Opportunity rejected, Opportunity unpublished
Internship Announcement Management: for handling internship announcement	Pub: Announcement published Announcement unpublished
Notification: for sending message to relevant users	Sub: Opportunity published, Opportunity rejected, Opportunity unpublished, Application rejected, SSI certificate loaded, Need update Announcement published, Announcement unpublished
Evaluation: for evaluating of summer internship	Sub: Summer practice finished Pub: Need update
Authentication activities are not handled in this case study, a third-party application can be used for authentication and authorization.	

Table 4.5. Undergrad Team's Microservices and related events of Summer Internship

Undergrad Team Microservices	**Undergrad Team Events**
Application: for handling application requests	Pub: Application file cannot uploaded
Announcement: for handling internship announcement and company opportunity requests	Pub: Announcement deleted, Announcement accepted, Announcement rejected
Announcement: for handling internship announcement and company opportunity requests	
Notification: for sending message to relevant users	Sub: Graded, Report rejected, Document evaluation process ended, Application file cannot uploaded
Evaluation: for evaluating of summer internship	Pub: Graded, Report rejected, Document evaluation process ended Sub: internship ended
Authentication: for handling authentication and registering activities	Pub: New company saved, Company accepted

CHAPTER 5

CONCLUSION

In this study, an event-driven analysis and design method for microservice-based systems called MicroArc is presented. The application of the proposed method and supporting tool are investigated in case studies (two exploratory and one explanatory). In this chapter, the results and findings of the study are discussed, the limitations of the proposed method, and directions for future works are presented.

5.1. Contributions

The goal of this study is to establish a method and supporting tool for the analysis and design of microservice-based systems in an event-driven manner. MicroArc method comprises modeling notations, a guiding process to articulate how the method is applied, and a supporting tool that enables modeling and transitions between models. It allows users to identify events that are produced or consumed by microservices from the business processes of an implemented software system and define candidate microservices based on high cohesion.

MicroArc allows transitions between models; therefore, integration between models can be established, and changes can be reflected in the connected models. Using this approach, developers can identify events and microservice candidates by modeling the flow of processes in the early phase of development.

The major contributions of this study are listed below:

- MicroArc method allows modeling a system from different perspectives and enables the identification microservices with related events
    - BMC helps identify the key activities related to the system.

  - EPC defines the flow of each key activity by illustrating events, activities (processes), and control points. Candidate microservices are identified based on aggregated activities.
  - MEFD shows interactions between microservices and illustrates event streams
- Since MicroArc method integrates models, any change in a model is reflected in other models.
- The supporting tool helps an analyst to model a system by using MicroArc method, and stores models in permanent storage.
- The supporting tool allows creating sub-processes in EPC. If a model is getting too large to easily view and difficult to follow the flow of process, activities can be divided into sub-processes to increase the readability of models.

The minor contributions achieved as part of this study are listed bellows:

- BMC model enables modeler to make an early size estimation on Key Activities.[27] It helps to make predictions about the duration and effort of a software system in the early stages.
- EPC model facilitates the estimation of the size of processes (activities).[28] It aids in estimating the effort required for the development phases of software systems.

5.2. Limitations and Future Work

The limitations of the study are given below:

- MicroArc method relies on Business Model Canvas (BMC) and Event-Driven Process Chain (EPC) methodologies. Using MicroArc method requires knowledge of these methodologies. A modeler should have basic foundations for BMC and EPC.
- Microservices based architecture has many challenges, like distributed system architectures. Microservices functionalities and related events may undergo changes to address challenges during the coding phase. However, these changes

may result in technical dept and models may not accurately reflect the real situations.

The future works related to the study are given below:

- The following functionalities can be included in supporting tools:
    - The tool can create code templates for microservices and export them to various coding environments.
    - The tool can create user stories from EPC diagrams and export them as backlog items to various project management tools.

REFERENCES

1. Sampaio, A. R.; Kadiyala, H.; Hu, B.; Steinbacher, J.; Erwin, T.; Rosa, N.; Beschastnikh, I.; Rubin, J. Supporting Microservice Evolution. *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)* **2017**. https://doi.org/10.1109/icsme.2017.63.
2. Thönes, J. Microservices. *IEEE Software* **2015**, *32* (1), 116–116. https://doi.org/10.1109/ms.2015.11.
3. Bellemare, A. *Building Event-Driven Microservices*; O'Reilly Media, 2020.
4. Dragoni, N.; Giallorenzo, S.; Lafuente, A. L.; Mazzara, M.; Montesi, F.; Mustafin, R.; Safina, L. Microservices: Yesterday, Today, and Tomorrow. *Present and Ulterior Software Engineering* **2017**, 195–216. https://doi.org/10.1007/978-3-319-67425-4_12.
5. Mamczur, P.; Mol, T. C. M.; Nowak, M. *State of Microservices*; Software House, 2020
6. Bilgin, B.; Unlu, H.; Demirörs, O. Analysis and Design of Microservices: Results from Turkey. *Turkish National Software Engineering Symposium (UYMS)* ***2020***, 1–6. https://doi.org/10.1109/UYMS50627.2020.9247022.
7. Unlu, H.; Tenekeci, S.; Yıldız, A.; Demirors, O. Event Oriented vs Object Oriented Analysis for Microservice Architecture: An Exploratory Case Study. *IEEE Xplore* **2021**. https://doi.org/10.1109/SEAA53835.2021.00038.
8. Brandolini, A. *Event Storming*; Leanpub, 2019.
9. Hevner, A. R.; March, S. T.; Park, J.-S.; Ram, S. Design Science in Information Systems Research. *Management Information Systems Quarterly* **2004**, *28* (1), 75–105. https://doi.org/10.5555/2017212.2017217.
10. Offermann, P.; Levina, O.; Schönherr, M.; Bub, U. Outline of a Design Science Research Process. *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology - DESRIST '09* **2009**. https://doi.org/10.1145/1555619.1555629.
11. Martin, Robert C. *Agile Software Development*; Pearson Education, 2003.
12. Bellemare, A. *Building an Event-Driven Data Mesh*; O'Reilly Media Inc, 2023.

13. Woolf, B. Event-Driven Architecture and Service-Oriented Architecture. *OOPSLA 2006: Workshop on Event Driven Architecture,* 2006. https://research.ibm.com/haifa/Workshops/oopsla2006/present/w06_eda_woolf.pdf (accessed 2024-10-01).
14. Papazoglou, M. P. Service-oriented computing: concepts, characteristics and directions. *Proceedings of the Fourth International Conference on Web Information Systems Engineering* **2003**, 3-12. https://doi.org/10.1145/944217.944233.
15. Evans, E. *Domain-Driven Design Reference: Definition and Pattern Summaries*; Domain Languages Inc, 2015.
16. Li, S.; Zhang, H.; Jia, Z.; Li, Z.; Zhang, C.; Li, J.; Gao, Q.; Ge, J.; Shan, Z. A Dataflow-Driven Approach to Identifying Microservices from Monolithic Applications. Journal of Systems and Software **2019**, 157, 110380. https://doi.org/10.1016/j.jss.2019.07.008.
17. Kamimura, M.; Yano, K.; Hatano, T.; Matsuo, A. Extracting Candidates of Microservices from Monolithic Application Code. *IEEE Xplore* **2018**. https://doi.org/10.1109/APSEC.2018.00072.
18. Al-Debagy, O.; Martinek, P. A New Decomposition Method for Designing Microservices. *Periodica Polytechnica Electrical Engineering and Computer Science* **2019**, *63* (4), 274–281. https://doi.org/10.3311/ppee.13925.
19. Ma, S.; Fan, C.-Y.; Chuang, Y.; Liu, I-Hsiu.; Lan, C.-W. Graph-Based and Scenario-Driven Microservice Analysis, Retrieval, and Testing. *Future Generation Computer Systems 100 (November)* **2019**, 724–735. https://doi.org/10.1016/j.future.2019.05.048.
20. Santos, N.; Salgado, C. E.; Morais, F.; Nascimento, M.; Silva, S.; Martins, R.; Pereira, M.; Rodrigues, H.; Machado, R. J.; Ferreira, N.; Pereira, M. A Logical Architecture Design Method for Microservices Architectures. *ECSA 2019 Companion Proceedings* **2019**. https://doi.org/10.1145/3344948.3344991.
21. Baresi, L.; Garriga, M.; De Renzis, A. Microservices Identification through Interface Analysis. *Service-Oriented and Cloud Computing* **2017**, 19–33. https://doi.org/10.1007/978-3-319-67262-5_2.

22. Gysel, M.; Lukas Kölbener; Giersche, W. I.; Zimmermann, O. Service Cutter: A Systematic Approach to Service Decomposition. *Lecture Notes in Computer Science* **2016**, 185–200. https://doi.org/10.1007/978-3-319-44482-6_12.

23. Bajaj, D.; Goel, A.; Gupta, S. C. GreenMicro: Identifying Microservices from Use Cases in Greenfield Development. *IEEE Access* **2022**, *10*, 67008–67018. https://doi.org/10.1109/access.2022.3182495.

24. Osterwalder, A. The Business Model Ontology: A Proposition in a Design Science Approach. Ph.D. Thesis, University of Lausanne, 2004.

25. Evans, E. *Domain-Driven Design : Tackling Complexity in the Heart of Software*; Addison-Wesley, 2014.

26. Scheer, A.-W.; Nüttgens, M. ARIS Architecture and Reference Models for Business Process Management. *Lecture Notes in Computer Science* **2000**, 376–389. https://doi.org/10.1007/3-540-45594-9_24.

27. Yıldız, A.; Demirörs, O. Size Estimation From a Business Idea. *Turkish National Software Engineering Symposium UYMS* **2022**, 89–99.

28. Kaya, M.;Demirörs, O. E-Cosmic: A Business Process Model Based Functional Size Estimation Approach. *37th EUROMICRO Conference on Software Engineering and Advanced Applications* **2011**, 404-410. https://doi.org/10.1109/seaa.2011.60.

APPENDIX A

HOW TO USE MICROARC TOOL

1. Click “Blank Project” to create a new project.

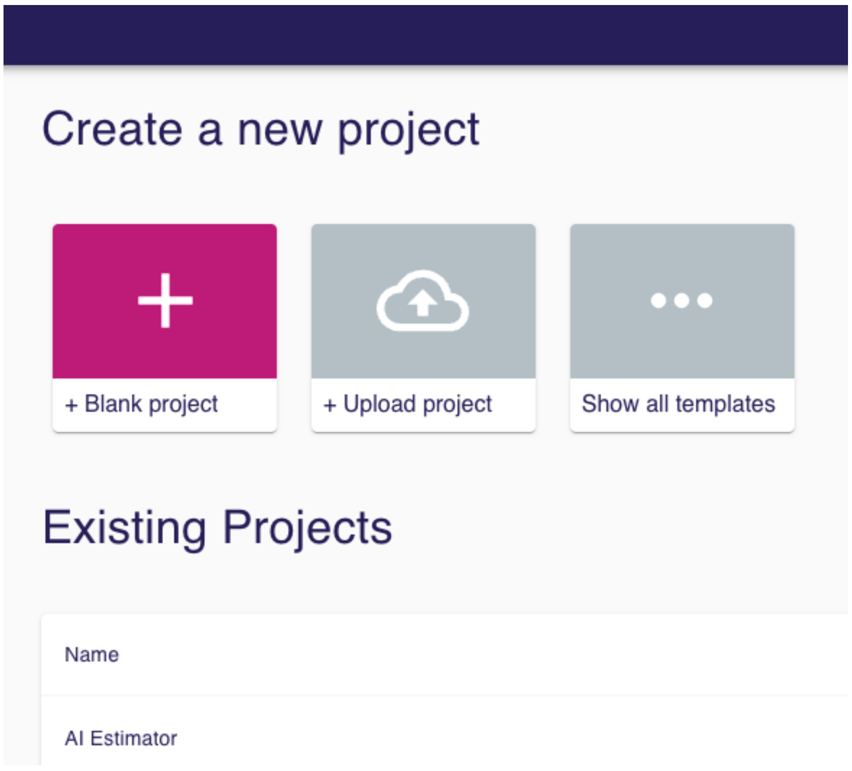
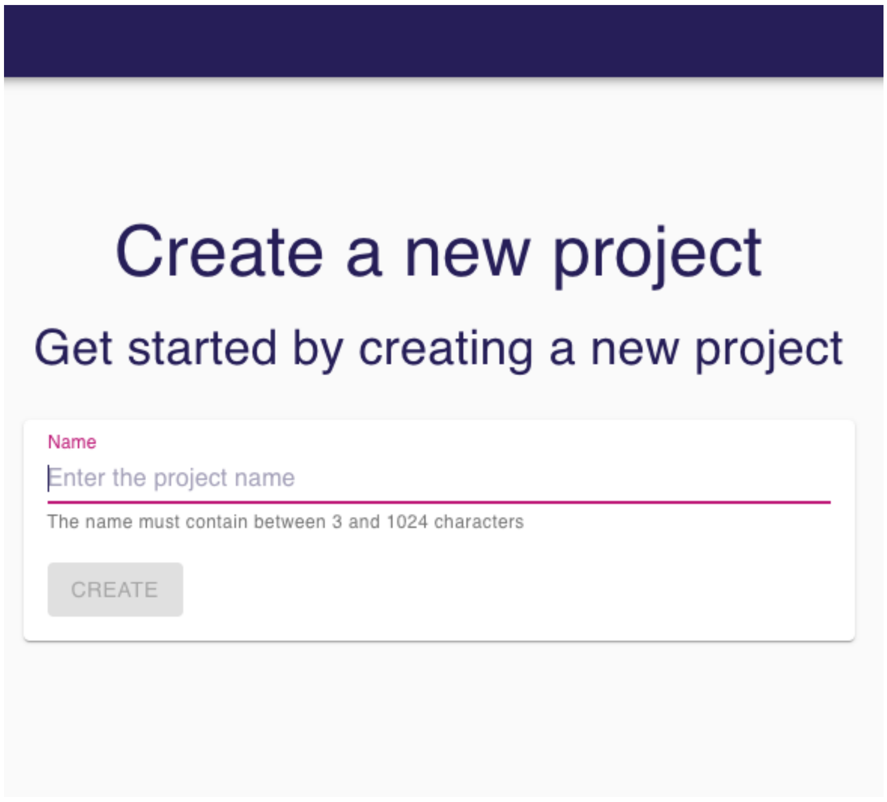


Figure A.1. Create blank project

2. Enter the project name then click “Create”.


Create a new project

Get started by creating a new project

Name

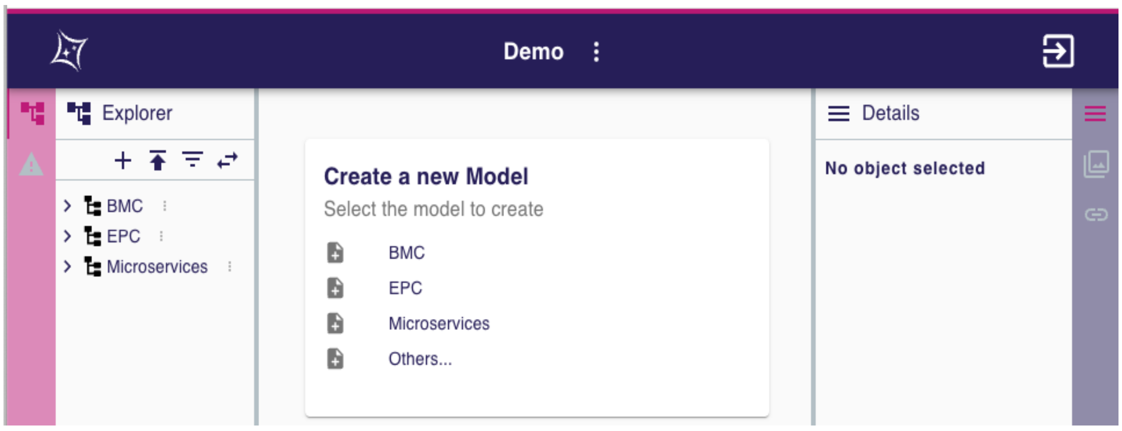
Enter the project name

The name must contain between 3 and 1024 characters

CREATE

Figure A.2. Create new project

3. Click “BMC” in order to create a new BMC model.


Demo

Explorer

BMC

EPC

Microservices

Create a new Model

Select the model to create

BMC

EPC

Microservices

Others...

Details

No object selected

Figure A.3. Create BMC

4. Click “BMC Model Diagram” in order to create a new BMC Model Diagram.

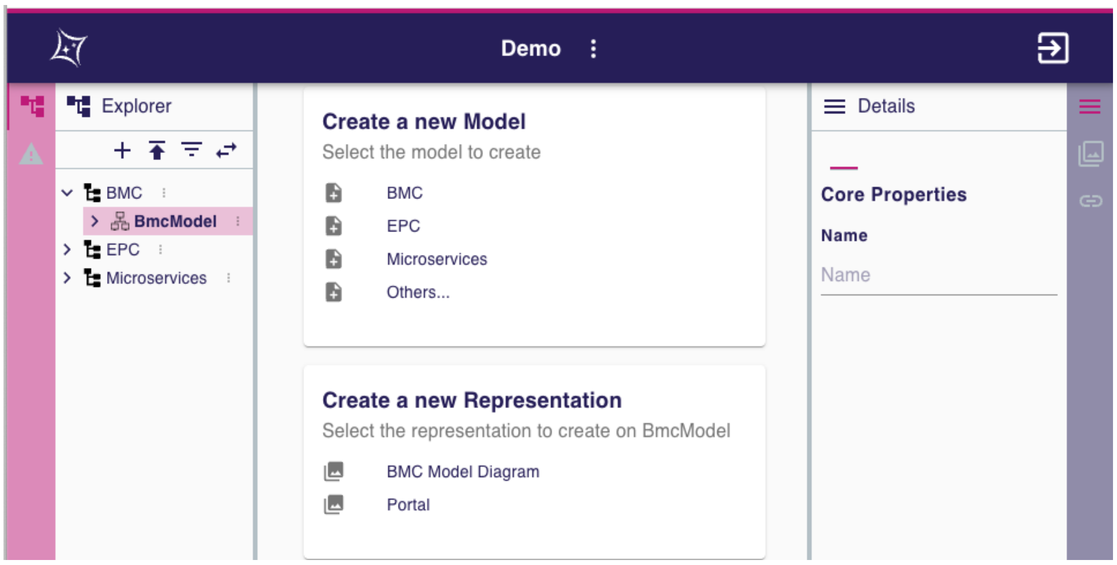


Figure A.4. Create BMC model

5. Click “BMC Model Diagram” in Explorer View to open Model View and start modeling.

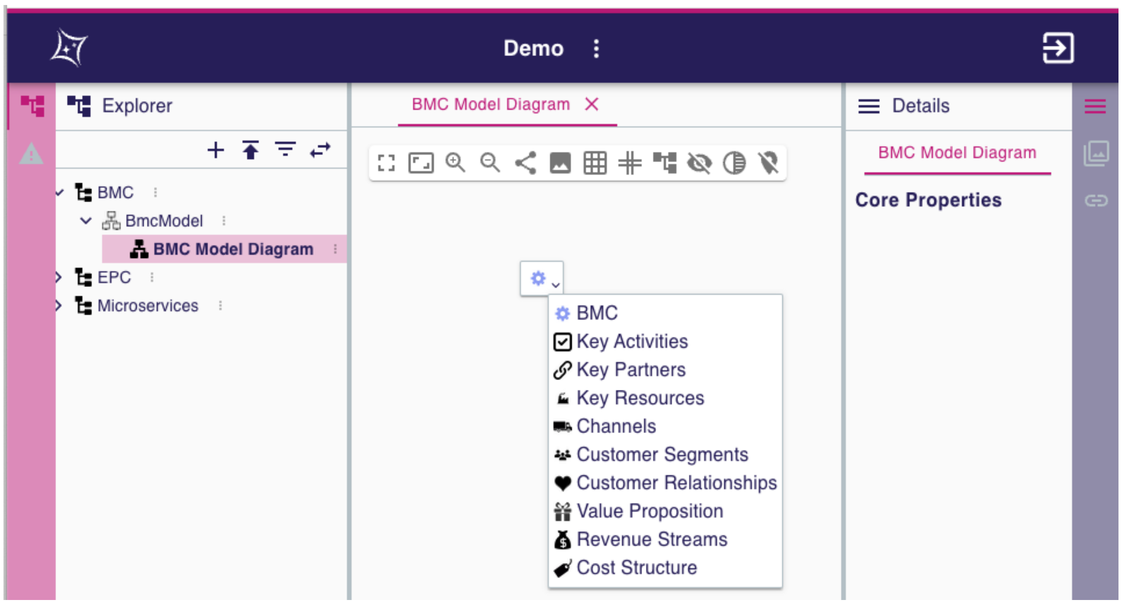
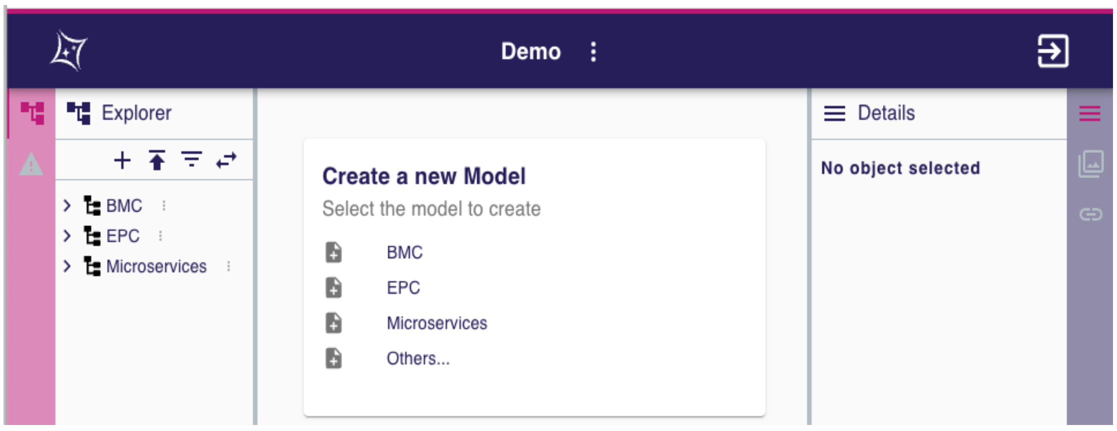


Figure A.5. BMC model diagram

6. Click “EPC” in order to create EPC model.



Create a new Model

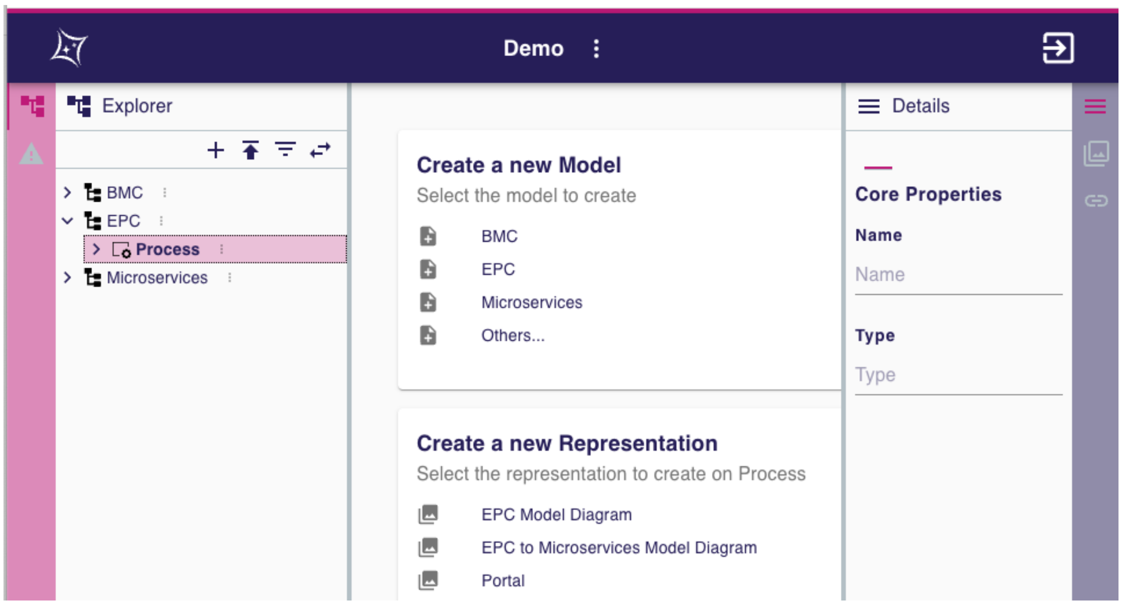
Select the model to create

- BMC
- EPC
- Microservices
- Others...

No object selected

Figure A.6. Create EPC

7. Click "EPC Model Diagram" in order to create a new EPC Model Diagram.



Create a new Model

Select the model to create

- BMC
- EPC
- Microservices
- Others...

Create a new Representation

Select the representation to create on Process

- EPC Model Diagram
- EPC to Microservices Model Diagram
- Portal

Core Properties

Name: Name

Type: Type

Figure A.7. Create EPC Model

8. Click “EPC Model Diagram” in Explorer View to open Model View and start modeling.

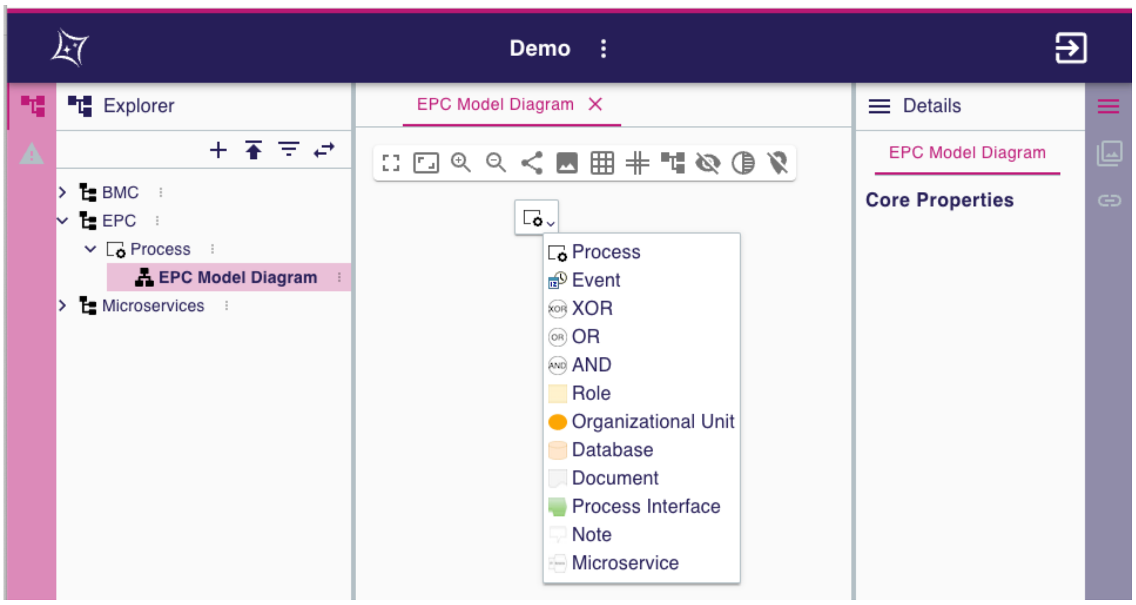
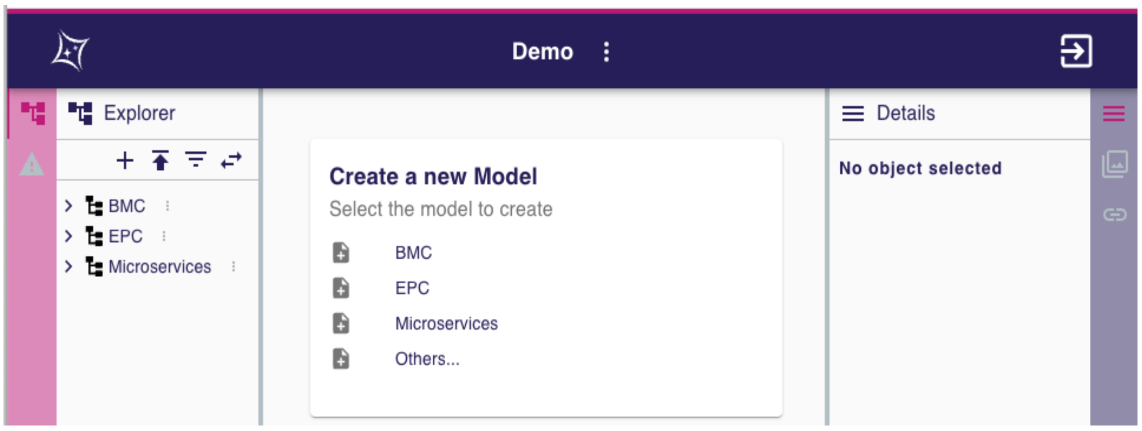


Figure A.8. EPC model diagram

9. Click Microservices” in order to create Microservice Event Flow Diagram.



Create a new Model

Select the model to create

BMC

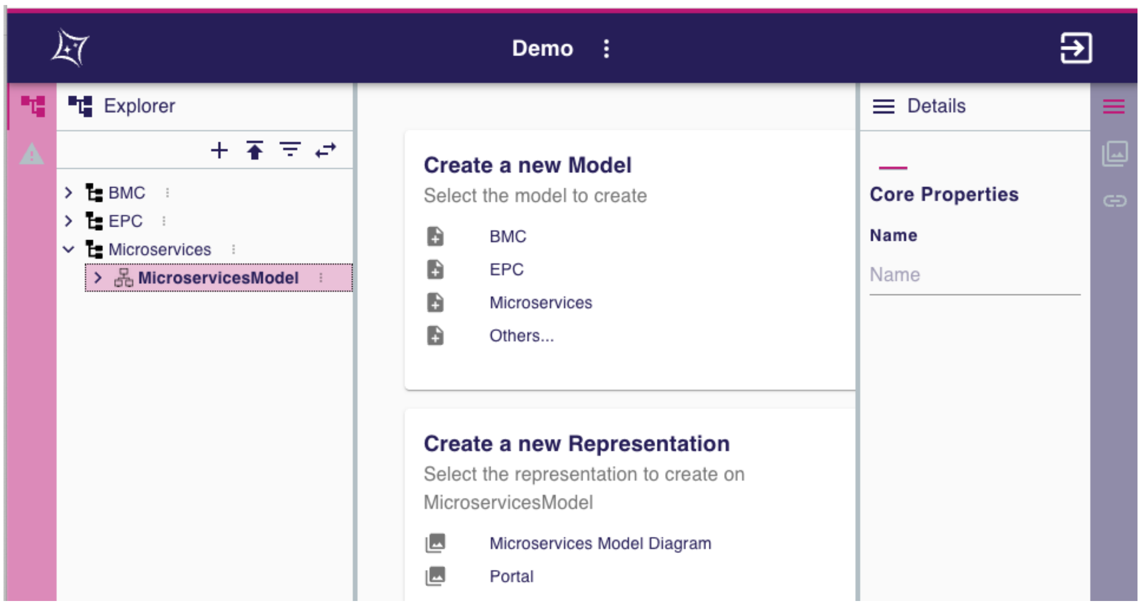
EPC

Microservices

Others...

Figure A.9. Create MEFD

10. Click "Microservices Model Diagram" in order to create a new Microservices Model Diagram.



Create a new Model

Select the model to create

BMC

EPC

Microservices

Others...

Create a new Representation

Select the representation to create on MicroservicesModel

Microservices Model Diagram

Portal

Figure A.10. Create Microservices model

11. Click "Microservices Model Diagram" in Explorer View to open Model View and start modeling.

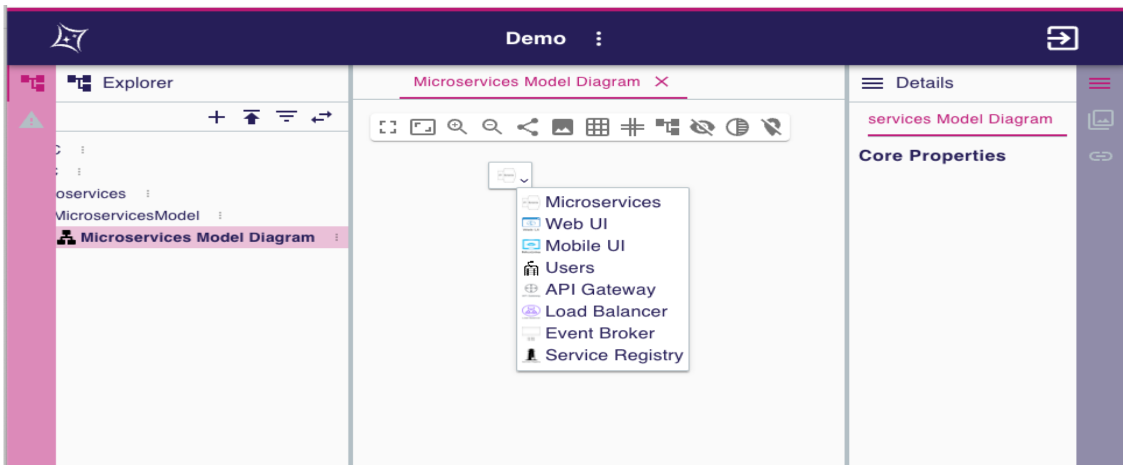


Figure A.11. Microservices model diagram

VITA

Ali YILDIZ

Education

- **M.Sc.:** 1994-1999, Dokuz Eylül University, Faculty of Engineering, Computer Engineering
  Thesis Title: "A Dashboard for personal software process improvement"
  Advisor: Assoc. Prof. Dr. Onur DEMİRÖRS
- **B.Sc.:** 1984-1988, Dokuz Eylül University, Faculty of Engineering, Electronic Engineering

Academic Experience

- 1997- 1999, Dokuz Eylül University, Research Assistant

Work Experience

- 1999-Continue, Bilgi Grubu, Software Engineer

Publications

- **Yıldız**, A., Demirörs, O. 2024. "MicroArc: Event-Driven Analysis and Design Method for Microservices." In *International Conference on Industry Sciences and Computer Science Innovation (iSCSi)*, Porto, Portugal.
- Hacaloğlu, T., Ünlü, H., **Yıldız**, A., Demirörs, O. 2024. "Software Size Measurement: Bridging Research and Practice." *IEEE Software* 41 (3): 49–58. https://doi.org/10.1109/MS.2024.3358079.
- **Yıldız**, A., Demirörs, O. 2022. "Size Estimation From a Business Idea." In *Turkish National Software Engineering Symposium (UYMS 2022)*, Erzurum.
- Unlu, H., **Yıldız**, A., Demirörs, O. 2022. "Effort Prediction with Limited Data: A Case Study for Data Warehouse Projects." In *48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 233–238. Gran Canaria, Spain. https://doi.org/10.1109/SEAA56994.2022.00044.