An Approach for Identifying Microservices using Clustering on Control Flow and Data Flow

Bachelor Thesis of

Niko Benkler

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Prof. Dr. Ralf H. Reussner
Second reviewer:	Jun.-Prof. Dr.-Ing. Anne Koziolek
Advisor:	Dr. Robert Heinrich

01. January 2019 – 31. April 2019

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE

. . . .

(Niko Benkler)

Abstract

Powered by the rise of cloud computing, agile development, DevOps and continuous deployment strategies, the microservice architectural pattern emerged as an alternative to monolithic software design. Microservices, as a suite of independent, highly cohesive and loosely coupled services, overcome the shortcoming of centralized monolithic architectures. Therefore, prominent companies recently migrated their monolithic legacy applications successfully to microservice-based architecture. They key challenge is to find an appropriate partition of legacy applications - namely *microservice identification*. So far, the identification process is done intuitively based on the experience of system architects and software engineers, mainly by virtue of missing formal approaches and a lack of automated tool support.
However, when application grow in size and become progressively complex, it is quite demanding to decompose the system in appropriate microservices. To tackle this challenge, the thesis provides a formal, graph-based identification approach using clustering techniques. Based on the business point of view, the approach identifies structural dependencies and data object dependencies to build a weighted graph. Clustering tools identify clusters that correspond to possible microservice candidates. To evaluate the quality and the effect of the process, the approach is applied to a case study that has already been decomposed into several microservices.

Zusammenfassung

Angetrieben durch den Aufstieg von Cloud Computing, agilen Entwicklungsmethoden, DevOps und Continuous Deployment Strategien etablierte sich die Microservice Architektur als Alternative zum monolithischen Software Design. Microservices sind eine Ansammlung unabhängiger, in sich zusammenhängende, aber lose gekoppelter Services, die die Defizite zentralisierter, monolithischer Software bewältigen. Namhafte Unternehmen hatten erst kürzlich ihre monolithische Alt-Software in ein microservice-basiertes System überführt. Eine Schlüsselaufgabe dabei ist es, die richtige Aufteilung der Alt-Software zu finden. Dieser Prozess wird Microserviceidentifikation genannt. Bis jetzt wurde er weitestgehend intuitiv und auf Basis von Expertenwissen durchgeführt. Der Hauptgrund dafür liegt vor allem in fehlenden formalen Ansätzen und automatisierter Unterstützung durch Software.
Dennoch wachsen Applikation mit der Zeit und werden zunehmend komplexer, sodass die Aufteilung eines Systems in dieser Komplexität durchaus herausfordern ist. Die Thesis stellt daher einen formalen, graph-basierten Ansatz vor, der mittels Clustering-Techniken mögliche Microservices extrahiert. Der Ansatz basiert auf der Prozesssicht und identifiziert strukturelle- und datenobjektbasierte Abhängigkeiten, um daraus einen gewichteten Graphen zu erstellen. Basierend auf dem zuvor erstelltem Graph ermitteln ermitteln Programme Cluster, die je einem Microservice entsprechen. Um die Qualität und Wirkungsweise des Ansatzes zu überprüfen, wird er auf eine Fallstudie angewendet, welche bereits in Rahmen anderer Arbeiten in Microservices gegliedert wurde.

Contents

List of Figures

List of Tables

1. Introduction

The monolithic software architecture is the traditional pattern to design software, where functionality is bundled in one single, large application [11]. Although monoliths have their strength, like fast development and simple deployment, they become an obstacle when they grow in size and become more complex [34]. Incomprehensible code structure makes it difficult to add functionality, fix bugs and enable new software engineering approaches like Continuous Delivery and Continuous Deployment[35]. Besides, the rise of cloud computing demands a new architecture that can fully exploit the rich set of features given by the cloud infrastructure [27].

Inspired by service-oriented computing, the microservice Architecture is about to become a promising alternative to overcome the shortcomings of centralized, monolithic architectures and consequently gains popularity in both, academia and industry [3]. Benefits like the increase of agility, resilience or scalability [37], the ability to use different technology stacks and independent deployment [4] and the efficient resource utilization in cloud environments [27] explain the usage of microservice-based application by big companies like Google, Netflix, Amazon, eBay [11] and Uber [37].

This thesis describes the current state of the art regarding microservices extraction and provides a systematic approach to decompose a system into microservices.

1.1. Motivation

Monolithic software applications develop over time and become more and more complex. The software structure becomes highly coupled and hard to maintain [17]. To tackle this issues, software engineers started to decompose their system into modules and provide the functionality over the network as Web Services [19]. The so-called *Service-oriented Architecture* (SOA) provides logical boundaries between the different software modules to address the design challenge of distributed systems. Nevertheless, Baresi et. al state that the boundaries between modules in SOA are too flexible and the application results in "a big ball of mud" [4]. Microservices make these boundaries physical as each service runs in its own process and only communicates with other services through well-defined lightweight mechanisms like REST [37]. Chen et al. consider the microservice architecture as a particular approach for SOA [11]. Others look at it as an evolution of SOA with differences in service reuse [4] or consider it to be the "contemporary incarnation of SOA" combined with modern software engineering practices like Continuous Deployment [19]. There is no consensus about the relationship between microservices and SOA but they both share common characteristics. The microservice architecture has many advantages over the monolithic style. Sec.2.2 elaborates the main aspects of microservices, including several benefits. Netflix, for instance, is able to cope with one billion calls a day to its video streaming API, by migrating their monolithic system to a high flexi-

ble, maintainable and scalable microservice architecture [11]. Consequently, moving existing applications to a microservice landscape is a upcoming philosophy in academia and industry [3].
Nevertheless, decomposing a system in loosely coupled, fine-grained and independent microservices is a time consuming task that requires tedious manual effort [19] and is technically cumbersome [14]. So far, it is done mainly intuitively and relies on the experience of software architects and system designers. Hence, a formal approach to identify microservices is required.

1.2. Research Questions and Contributions

The microservice architecture is a fast rising approach to structure a system in high cohesive but loosely coupled and independent services. Many companies like Amazon, migrated their monolithic legacy software to microservice in order to fully leverage the benefits of cloud computing and new software engineering approaches like Continuous Deployment [27]. Large applications are decomposed into small, independent microservices where each service can be independently scaled and deployed.
However, one of the biggest problem in designing a microservice architecture is to decompose a monolithic application into a suite of small services while keeping them loosely coupled and high cohesive. This challenging task is also known as *microservice identification* [3].
Baresi et al. state that a "proper" microservice identification defines how a system will be able to evolve and scale [4]. Others claim, that finding the optimal microservice boundaries [18] and service granularity [20] is the key design decision to fully leverage the benefits of microservices.
So far, the partition is performed mainly intuitively based on the experience and know-how of experts that perform the extraction. Hassan et al. criticises a lack of systematic approaches to reduce the complexity of the extraction process [20]. Extracting microservices from monoliths therefore requires tedious manual effort and can be very costly [37] [30]. This leads to the following Research Question (RQ):

> **RQ1: Which is the most appropriate strategy to decompose a system into microservices?**
>
> To identify possible strategies, an extensive literature research is conducted. Suitable strategies and approaches are compared based on criteria identified in the literature research.
>
> **RQ2: What formal approach can be constructed to identify possible microservices without detailed know-how and manual effort?**
>
> To that end, the most promising strategy identified in RQ1 is used as basis. Thereupon, a formal approach is elaborated that aims to reduce the complexity and manual work that has to be done when identifying microservices.

RQ3: What is the accuracy of the approach?

Research question RQ3 is tackled by applying the approach to the Common Component Modelling Example (*CoCoME*). The subsequent system decomposition is evaluated by comparing the identified microservices with two other approaches: First, Tyszberowicz et al. [37] provide a decomposition of CoCoME based on their approach. Second, we identified and implemented a microservice-based version of CoCoME manually.

1.3. Thesis Outline

The proposal is structured as follows:

- Chapter 2 presents the background information on monolithic software architecture and microservice-based architecture. For the latter one, benefits and challenges are elaborated.
- Chapter 3 introduces the common case study *CoCoME* that is used to apply and evaluate the approach. Special attention is given to the system specifications.
- Chapter 4 outlines the current state of the art concerning microservice identification. First, the process of literature review is presented. Secondly, the most promising strategies and approaches are described and further compared using adequate criteria.
- Chapter **??** sketches a first attempt for a formal approach, that is further elaborated during the ongoing work on this thesis.
- Chapter 8 explains how the elaborated approach can be applied and evaluated using the case study *CoCoME*
- The proposal is concluded by chapter **??**, where a schedule and accompanying milestones are defined.

2. Background

2.1. Monolithic Software Architecture

The monolithic software architecture is a well-known and the most widely used pattern for Enterprise Applications, which usually are built in three main parts (top to bottom): The client-side user interface (Tier 3), the server-side application that contains the entire business logic (Tier 2) and the persistence layer handling the database access (Tier 1). Fig. 2.1 illustrates the architectural difference between a standard three tier application and a exemplary microservice-based architecture. The server-side application - *the monolith* - is a single unit and deployed on one application server [34]. The software structure, if well defined, is composed of self-contained modules (i.e. software components), where each module consists of a set of functions [14]. The monolith implements a complex domain model, including all functions, many domain entities and their relationships. For small applications, this approach works relatively well. They are simple to develop, test and deploy [37]. Fast prototyping is supported by the current frameworks and development environments (IDE), which are still oriented around developing single applications [34].
But once they grow in size, they become exceedingly difficult to understand and hard to maintain without reasonable effort [37] [18]. A complex and large code base prevents a fast addition of new features and makes the application risky and expensive to evolve [26]. Alterations to the system, even though they might be small, result in a redeployment of the whole monolith application due to its nature being a single unit [37]. Moreover, it is difficult to adopt newer technologies without rewriting the whole application, as monolith are build on a specific technology stack [34] [30].
Scaling is only possible by duplicating the entire application, namely *horizontal scaling*. Consequently, large portions of the infrastructure remains unused, if only parts of the application need to be upscaled or even used [23] [17].
Chen et al. provide a short résumé:

> *"Successful applications are always growing in size and will eventually become a monstrous monolith after a few years. Once this happens, disadvantages of the monolithic architecture will outweigh its advantages."*
>
> - Rui Chen [11]

2.2. Microservices

"The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API."

- Martin Fowler [16]

2.2.1. Definition

The above quotation is a widely adopted definition of the term *Microservice*, provided by M. Fowler and J. Lewis [16], the pioneers of the microservice architecture. However, the term is not formally defined. Amiri et al. describes microservices as a collection of cohesive and loosely coupled components, where each service implements a business capability. The author introduces three principles upon which the architecture is build: *Bounded Context, Size, Independence* [3].

The first principle is about related functionality, that is combined in a single business capability - the *bounded context* [37]. Each capability is implemented by one microservice. The *Size* of a microservice is defined by the number of features it provides (namely bundled functional capabilities) [24]. There is no consensus about the "proper" size of a microservice [33], but several guidelines exists: Services should focus on one business capability only [3]. Others state, that the size of a microservice should not exceed a level, where it cannot be rewritten within six weeks [24]. However, the sizes vary from system to system [37] and even different sizes for each microservice in a specific system are possible [33]. The bottom line of *Independence* is in Amiri's description of microservices as "a collection of high cohesive and loosely coupled components" [3]. High cohesive services implement a relatively independent piece of business logic (at the most one business capability). Further, microservices should hardly depend on each other, which is the idea of being loosely coupled [11].

Communication between microservices is achieved by lightweight message passing mechanisms such as *REST*. Each service exposes a well defined interface (*API*) with endpoints that provide information using standard data formats [37]. The design of microservices mainly follows the *Single Responsibility Principle (SRP)*: Each service should not have more than one reason to change [15]. The SRP mainly corresponds to the idea of not implementing more than one business capability. The following covers the benefits and challenges of the microservice architecture.

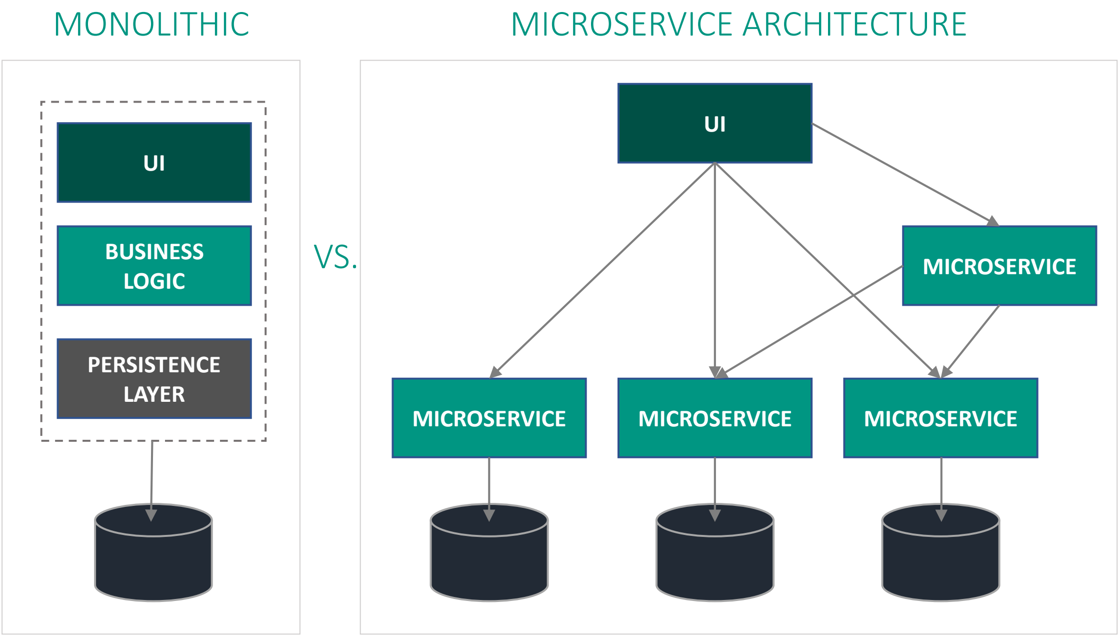

Figure 2.1.: Monolithic vs. Microservice Architecture

2.2.2. Benefits

Fast and Independent Deployment

As a matter of fact, each microservice is deployed independently [4]. Changes to the code do not result in a full redeployment of the entire application [37]. Consequently, software developers are able to react much quicker to changes in business requirements. This includes an acceleration in error correction. Per contra, any changes in a monolithic code base requires a time consuming build of a new version and the redeployment of the entire application [16].

Availability, Resilience and Fault Isolation

Microservices are designed to operate independently of each other and to tolerate failure of services [16]. Large parts of the application remain unaffected of partly failures and the availability of the system is, at least partly guaranteed. Monolithic application do not provide this type of fault isolation. If a failure occurs, the whole application remains unavailable as it is usually running in a single process [30].

Scalability and Resource Utilisation

Small and independent microservices allow more fine-granular horizontal scaling [24]. Single services can be duplicated to cope with changing workload during runtime [11]. Thus, dynamic (de-)allocation of resources on demand prevent infrastructure from being idle [14]. Scaling monoliths can only be attained by duplicating the entire application , leaving resources unused [18]. Further, each microservice is deployed on the best suitable infrastructure for its needs, allowing a more efficient system organization [34].

Improved Productivity

In traditional software development, teams are divided based on their expertise: Database architects, UI-developers and server-side engineers, resulting in a three-tiered application (cf. Sec.2.1). Additionally, software engineers are responsible for the development only. Deployment is part of the operations team. This team structure results in high communication overhead and slows down the productivity [31].

In contrast, microservices are organized around business capabilities and require cross-functional, independent teams [3]. Each team has the full range of skills required for the end-to-end realization of a microservice, including UI-development, database architecture, back-end engineers and project management. This minimizes the communication and interaction between the teams and thus, speeds up the productivity. Ultimately, microservices enable a more agile flow of development and operation [18], also referred as *DevOps*.

Neutral development technology
Microservices are highly decoupled from each other, as they use standardized and lightweight communication mechanisms such as REST [37]. Microservices can be realized using different programming languages, technologies and even deployment environments [11]. Developers are consequently not longer limited to use a single technology for the whole application. They can choose the most appropriate technology for each particular business problem or try out some new technology without rewriting the whole application [19] [26].

2.2.3. Challenges

The previous section provides a vast amount of benefits that come with microservices. However, it is not the panacea of software engineering and has to face some challenges before being able to fully benefit from them. The challenges are further described in the following.

Expensive Communication
Microservice use network protocols such as *HTTP* to communicate with each other. Compared to standard, inter process communication (*IPC*) as used in monoliths, remote procedure calls a more expensive [2]. As a consequence, applications experience a decrease in performance as network communication is generally slower than IPC.

Technical Challenges
Microservices require a high degree of infrastructure automation [17]. The benefits of fast and independent deployment cannot be utilized, if it has to be done manually. Dynamic (de-)allocation of resources when scaling individual microservices need a well defined and structured cloud environment [27]. Besides, the distributed microservice landscape complicates the logging mechanisms and performance monitoring [2]. Traditional centralized logging, as it is used in monolithic applications, is not longer applicable. Instead, a careful aggregation system to gather logging and monitoring data from each service is required.

Organizational Challenges
The microservice approach needs the establishment of cross-functional teams [16]. Adopting *Continuous Practices*, such as *Continuous Deployment*, are essential for the success of a profitable microservice architecture. Therefore, closer collaboration between development teams, operational staff and management has to be established. In summary, a costly and time consuming restructuring process of the entire organization is required [7].

Data Consistency
Distributed systems need to share data. Heinrich et al. propose two concepts for the database

architecture [37]: The first concept applies the basic idea of the microservice approach, as it splits the database into several parts. Each microservice has its own database which manages the entities that belong to the corresponding bounded context. Higher speed and horizontal scaling are facing data consistency issues. Data needs to be synchronized which leads to inconsistency, if services are unavailable. The second concept is about sharing a single database. This approach overcomes the issue of consistency, as data is stored centrally. But sharing results in a loss of independence. Scaling can only be achieved through replicating the whole database. Research revealed, that the first concept is preferred [37].

Decomposition

Decomposing a system into microservice is a very complex task that requires experienced system architects and domain experts [16]. Identifying the right granularity of microservice is one of the key issues. Too fine grained services cause inefficiency due to a high amount of expensive inter-service calls [33]. Developing the basic communication infrastructure adds additional complexity and slows down the initial developing process [34].

3. Case Study

The *Common Component Modelling Example (CoCoME)* is a case study on software architecture modelling [22][21]. In this thesis, it is used to demonstrate and validate the presented approach. Sec.3.1 provides a short introduction of the demonstrator, followed by a presentation of its system specifications.

3.1. Introduction to CoCoME

CoCoME represents a trading system as it can be found in a supermarket chain. The main task is handling and processing sales at a single store of the chain. Therefore, customers can pick goods and place them on the *Cash Desk* whose main component is a *Cash Desk PC*. Several other components like *Bar Code Scanner*, *Light Display*, *Printer*, *Card Reader* and *Cash Box* are wired by the *Cash Desk PC*.

Multiple *Cash Desks* of a single store form a *Cash Desk Line*, which is connected to the *Store Server*. A set of stores in the CoCoME chain is organized as an enterprise where each store is connected to a single enterprise server.

More detailed description of the CoCoME system can be found in [22][21]. The next section provides information about the system requirements specifications in form of use cases.

3.2. System Specifications

The system specification is informal and given in the form of detailed use cases. Fig.3.1 provides an overview of the use case of CoCoME. A full detailed description can be found in [22].

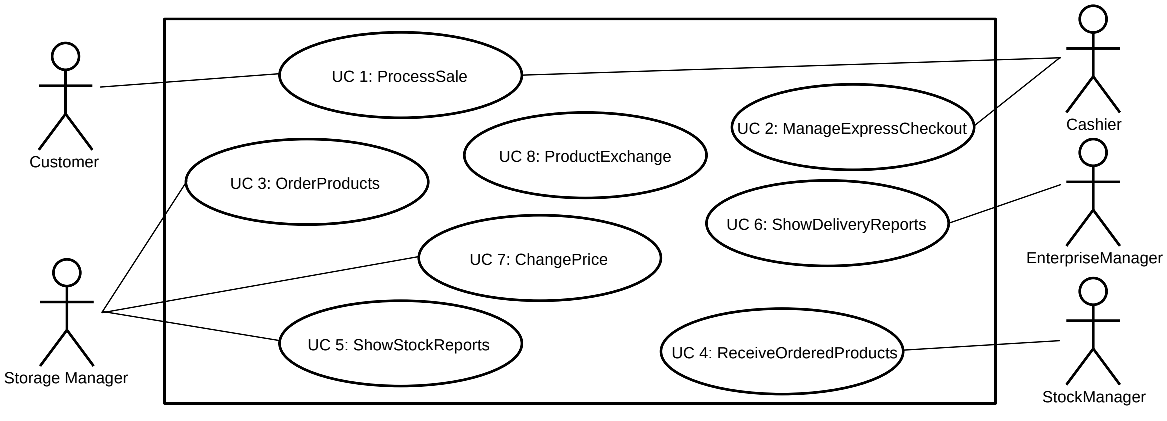

Figure 3.1.: Use Case Diagram of CoCoME

Use case description

- *Process Sale:* Handles the products a customer wants to purchase and the payment (either cash or card).
- *Manage Express Checkout:* The cash desk switches automatically in the express mode (under certain conditions). The cashier is able to switch back in normal mode.
- *Order Products:* A store manager can order products from suppliers.
- *Receiver Ordered Products:* Ordered products which arrive at the store need to be checked for correctness and inventoried by the stock manager.
- *Show Stock Reports:* A store manager can request a stock-related report for his/her store.
- *Show Delivery Reports:* Calculation of the mean time for a delivery.
- *Change Price:* The sale price of a product is changed.
- *Product Exchange:* Automatic stock exchange if a store is running out of stock and other stores still have the required product

4. State of the Art

This chapter outlines the current state of the art regarding microservice identification. Sec. 4.1 presents the search strategy and several existing approaches (Table 4.1) that deal with the identification of microservices. Thereupon, the approaches are further explained and finally compared on the basis of several criteria.

4.1. Literature Review

The approaches mentioned in table 4.1 are the result of an extensive literature research which was conducted using the digital libraries IEEE [1], ACM [2] and SpringerLink [3]. The web search engine Google Scholar [4] provided further approaches and general information.
"Identifying Microservices using Functional Decomposition" [37] was provided by the supervisor of this thesis. Besides, *"Service Cutter - A Systematic Approach for Service Decomposition"* [19] was cited by various approaches, including [4] while the remaining papers were found using following search string:

["identify" OR "identification" OR "migrating" OR "monolith" OR "decomposition" OR
"decompose monolith" OR "decompose"] AND "microservice"
OR
"microservice" AND ["identification" OR "transformation" OR "refactor"]

Table 4.1 presents the 8 most promising approaches in the area of microservice identification. Other papers like [38] only presented a conceptual train of thought, whereas [27], for instance, focuses on migrating strategies on infrastructural level. This thesis mainly focus on the identification part and disregards the actual implementation and deployment process afterwards. To compare the available approaches, criteria have to be defined. Sec.4.3 introduces 8 criteria and explains why they take part in the comparison. The comparison itself is done by applying the criteria to each approach using Table 4.2 and 4.3. Incidentally, the comparison including some criteria are inspired by the work of [18]. Further information is given in textual form in the same section.

[1] http://ieeexplore.ieee.org
[2] http://portal.acm.org
[3] http://www.springerlink.com
[4] http://scholar.google.com

Link	Titel	Author (Year)	Origin	Search String
[30]	Extraction of Microservices from Monolithic Software Architectures	G. Matzlami et. al. (2017)	Google Scholar	*microservice identification*
[3]	Object-Aware Identification of Microservice	M. J. Amiri (2018)	IEEE	*identification microservices*
[4]	Microservices Identification Through Interface Analysis	L. Baresi et. al. (2017)	SpringerLink	*microservice identification*
[37]	Identifying Microservices Using Functional Decomposition	S. Tyszberowicz et. al. (2018)	*provided*	*n/a*
[33]	Partitioning Microservices: A Domain Engineering Approach	I. J. Munezero et. al. (2018)	ACM	*partition microservices*
[11]	From Monolith to Microservices: A Dataflow-Driven Approach	R.Chen et. al	IEEE	monolith microservice
[14]	Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems	A. De Alwis et. al. (2018)	Google Scholar	identify microservices
[19]	Service Cutter: A Systematic Approach to Service Decomposition	M. Gysel et. al. (2016)	[4]	*n/a*

Table 4.1.: List of authors and approaches

4.2. Approaches for Identifying Microservices

The following section provides a short introduction in the approaches mentioned in table 4.1.

Extraction of Microservices from Monolithic Software Architectures
The approach presented in [30] is a class based extraction model, that uses (meta-)information of a version control system *(VCS)* such as Git[5] to identify microservices. The approach is divided in two phases: The *Construction Phase* and the *Clustering Phase.* Starting with a given code base, the approach uses three different coupling strategies and the information provided by the *VCS* to transform the monolith into a weighted graph. Here, the nodes represent classes, and the edges have weights according to the chosen coupling strategy. In the second phase, a clustering algorithm determines possible microservices (each cluster is a microservice candidate).

Object-Aware Identification of Microservice
[3] identifies microservices from business processes, using the widely known *Business Process and Model Notation (BPMN).* The approach uses clustering based on structural dependency and data object dependency. The first aspect is extracted from related activities within the business process model. A relation exists, if an edge directly connects a pair of activities or if only gateways are in between.
The latter aspect is based on the data object read and writes of each activity. Activities that are directly or indirectly connected and perform write or read operations are more likely to partition into the same microservice.
Both relations are stored in a separate matrix. To aggregate both relations, the matrices are summed up by simple matrix addition. Second to last, the relation matrix is transformed into a weighted graph using the values of the matrix as weights. Finally, clustering algorithms determine clusters that represent microservice candidates.
Microservices Identification Through Interface Analysis
In [4], the author proposes an approach that is based on semantic similarity of functionality specified through OpenApi[6] specifications (OpenApi defines a language-agnostic, standardized and machine-readable interface for RESTful APIs). The similarity depends on a reference vocabulary: each operation of the specification is analysed along with its resources (parameters, return values, complex types) and mapped to a concept of the chosen reference vocabulary. Each mapping has a score, based on a fitness function that uses the collocation of words (called terms) found in the operation and in the concepts. A co-occurrence matrix contains all mappings of possible pairs of terms and concepts. It is maximized to obtain the best mappings. Finally, this approach identifies potential candidate microservices, as fine-grained groups of operations, that are mapped to the same reference concept.

Identifying Microservices Using Functional Decomposition
The approach presented in [37] identifies microservices by functional decomposition of the software requirements, provided as use case specifications. In order to achieve the decompo-

[5] https://github.com/
[6] https://www.openapis.org/

sition, the system is modelled as a finite set of *system operations* and the system's *state space*. Use cases provide the necessary input data: Verbs found in the use cases serve as *system operations* and nouns correspond to the *state variables* that the operations read or write. The state variables constitute the state space. Relationships between the operations and the variables are stored in a relation table, that is visualized as a weighted graph. Finally, the approach uses graph analyse tools to determine clusters, where each cluster is a potential candidate of a microservice that fulfils the criteria of low coupling and high cohesion.

Partitioning Microservices: A Domain Engineering Approach
Munezaro et al. [33] propose an approach to identify appropriate microservices using *Domain-driven Design (DDD)* patterns. As a prerequisite, developers define a domain by using ubiquitous language. The domain indicates what the system does, precisely the system responsibilities, and what functionality it must implement. Domain experts define the boundaries of each responsibility and make it as a *business capability*, where a business capability is something that a system does in order to generate value. Each business capability is a microservice. When defining the boundaries, the focus is on the relationships among the services to minimize cross-cutting transactions.

From Monolith to Microservices: A Dataflow-Driven Approach
Chen et al. [11] uses a top-down data flow driven decomposition approach to determine high cohesive and loosely coupled microservices. Before the actual identification process starts, a *Data Flow Diagram (DFD)* needs to be constructed on the users' natural language description of the system to illustrate the detailed data flow. The first step of the approach consist of manually constructing a purified DFD, which focuses on data's semanteme and operations only. Afterwards, the purified DFD is algorithmically transformed into a decomposable DFD which is finally used to extract potential microservice candidates.

Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems
This is an approach that utilizes heuristics to specify two fundamental areas of microservice discovery: Function splitting based on common object subtypes and functional splitting based on common execution fragments across software [14].
The discovery process consists of two steps: First, the code, database tables and the SQL queries are evaluated to identify business objects and their relationships. Along with a set of given execution call graphs (different sequences of operations; generated though e.g. analysing log data), the information found is passed to the second part of the process. Algorithms processes the call graphs of the legacy system to derive a set of subgraphs and analyse which fragments are related to the same business objects in order to recommend possible microservices.

Service Cutter: A Systematic Approach to Service Decomposition
Gysel et al. [19] introduce a service decomposition tool based on 16 coupling criteria coming from industry and literature. A coupling criterion is a decision driver to decide whether data, operations or artifacts (generalized under the term *nanoentity*) should or should not be owned and exposed by the same service. Additionally, each criterion has a different score according to its priority. The input is in form of various *System Specification Artifacts (SSAs)*, such as domain models and use cases. The tool *Service Cutter* extracts coupling criteria information

out of it, that must be prioritised by a user. To analyse and process the coupling criteria, Service Cutter creates a weighted graph. The nodes represent the nanoentities and the weights on and edge is the sum of all scores per criterion, multiplied by a user defined priority. In the end, an exchangeable clustering algorithm identifies potential microservice candidates where each cluster correspond to a high cohesive and loosely coupled service.

4.3. Comparison

Table 4.2 and 4.3 provide a short description of the identified approaches mentioned above regarding some comparison criteria. The following criteria were used: **Basis Concept** recaptures the underlying approach of the microservice identification for classification purposes. **Prerequisites** presents the necessary preconditions for the success of the approach. For example, the approach mentioned in [30] cannot be used without meaningful VCS[7] data. The **Input** row describes the type and amount of input that is used realize the approach, i.e. Data Flow Diagrams in [11]. The row **Tool Support** indicates, whether the approach has been implemented or if other supporting tools are available to simplify the identification of high cohesive and loosely coupled microservices. The **Degree of human involvement** is part of the comparison, as this thesis aims to reduce the complexity of the service identification while keeping the required amount of expertise and manual tasks on a minimum. As evaluated in Sec.2.2, defining fine-grained microservices is a key challenge. Therefore, the approaches need to be compared in regard to the **Granularity of the recommended Microservices**. Some approaches allow an adjustable level of the granularity (e.g. [30]), whereas others generate a predefined granularity (e.g. always the most fine-grained microservice candidates [14]). **Validation** compares how the approaches are validated to strengthen the credibility of the individual results. Each approach might have some drawbacks regarding it's applicability to universal systems, required amount and type of input, user interaction and further expertise. **Limitations** is meant to point out the identified drawbacks. The following paragraph replenishes and explains the results given in Table 4.2 and Table 4.3 shortly:

The approach mentioned in [30] is the result of a master thesis [31]. Therefore, the degree of available information is larger compared to other approaches. The algorithmic recommendation of microservices candidates is implemented in a web-based, open source prototype and permits to choose three different coupling criteria, which can be combined for better results. Nevertheless, the main limitation relies in its type of input data: meaningful VCS data. For instance, developers must not commit changes on two independent functionalities together, but split it up. If this is not the case, the outcome may be wrong.

Amiri's approach [3] uses the open-source clustering software *Bunch*[8] . Further, information about the validation process (i.e. tested systems) are not given, but he claims that the process was successful. The weighting of the relationships lack formal explanation and need further analysis. Besides, the aggregation of structural dependency and data object dependency lack formal explanation too. Eventually, Amiri does no clearly differentiate data- and control flow.

[7] Version Control System

[8] https://www.cs.drexel.edu/ spiros/bunch/

Nonetheless, the approach is straightforward and does not require any user involvement once the input data is available.

Baresi et al. [4] developed an experimental prototype to validate their results. They used a multitude of specifications and compared the outcome with results of software engineers and the tool *Service Cutter* [19]. Nevertheless, the outcome highly depends on the chosen reference vocabulary and well-defined APIs in the legacy system. Operations and resources (variables, return values) have to be expressive and represent what they do. Variable names like *temp* or *var1* would result in a useless service decomposition.

[37] uses external tools to realize the approach. Once the operations and state variable are identified, the identification method is universally applicable to all sort of legacy systems and also greenfield applications[9]. Nonetheless, identifying relevant nouns and verbs that represent the operations and state variables is only partly supported by tools. It still requires human expertise to eliminate duplicates and identify ambiguities.

Munezero et al. present a conceptional approach based on DDD patterns.[10]. Although the domain-driven design approach is currently the most common technique for identifying microservices ([37][16][17] and more), it does not tackle RQ2 (precisely *RQ2*) mentioned in Sec.1.2, as it requires the expertise and experience of domain experts.

Chen's semi-automate approach [11] is based on Data-flow Diagrams (DFD). Transforming the traditional DFDs to purified DFDs is not trivial an therefore requires a vast amount of additional manual work. Nevertheless, the purified DFD represents the real information flow of the corresponding business logic in the legacy system and therefore provides valuable information regarding potential inter- and intra service communication.

The approach presented by Alwis et al. [14] is a more complex method for microservice identification: Many steps and prerequisites are necessary to prepare the input data. For example, expressive *Log Files* are required to generate call graphs. Further, source code and the system's database is required to identify so-called business objects and their relationships. The latter one lacks a formal description in the paper. Additionally, the algorithms to identify potential microservice candidates are solely provided conceptually without further tool support.

Service Cutter [19] is a mature open-source software with available wiki. It was the first attempt to automatize service extraction and is therefore a reference project for other approaches like [30] and [11]. It uses 16 coupling criteria extracted from industrial experience and knowledge to decompose a system into services. However, the input requires special formats and consequently extensive and time consuming preparation.

[9] Project which lacks any constraints imposed by legacy systems

[10] Domain-driven Design

Approach/Criterion	Mazlami et al. [30]	Amiri [3]	Baresi et al. [4]	Tyszberowicz et al. [37]
Basic Concept	meta-data aided graph clustering	business process oriented graph clustering	semantic similarity of OpenApi specification	functional decomposition of sw requirements
Prerequisites	applications with meaningful VCS data	business processes and entities available	well-defined Api with proper naming	specification of software requirements
Input	Source Code and VCS meta data	BPMN business processes with data object reads and writes	reference vocabulary (fitness function), OpenApi specifications	use cases
Tool support	prototype available (https://github.com/gmaz frontend)	Clustering tool "Bunch"	experimental prototype (https://github.com/mgarriga/decomposer)	use external graph visualize and analyse tools
Degree of human involvement	choose amount of clusters that will represent the microservices	no interaction needed	user defines level of hierachy	manual elimination of synonyms, irrelevant nouns and verbs
Granularity	depends on choosen amount of clusters	depends on iteration of genetic algorithm for convergence of fitness function	depends on choosen hierachy lebel, varies from one to many	depends on size of business capability
Validation	experiements using open-source projects with VCS data (200 to 25000 commits, 1000 to 500000 LOC, 5 to 200 authors)	multiple experiments, results compared with domain experts knowledge	452 OpenApi specification, 5 samples compared with results of sw-engineers and [19]	case study, compared to three manual implementations
Limitation	need meaningful VCS data and ORM model for its data entities	given weight definitions lack formal explanation	depends on reference vocabulary and well-defined interfaces	manual revision of operations (nouns) and state variable (verbs)

Table 4.2.: Comparison of Approaches, Part I

Approach/Criterion	Munezero et al. [33]	Chen et al. [11]	Alwis et al. [14]	Gysel et al. [19]
Basic Concept	define business capabilities by using domain-driven design patterns	algorithmic identification of microservices using data flows	graph-based identification process using heuristics to describe call graph similarities	service decomposition based on 16 coupling criteria
Prerequisites	domain defined by ubiquitous language	systems's data flows constructen on users' natural langugae description	Log files of legacy system	various System Specification Artifacts (SSAs) in specified format
Input	well defined domain model	Data Flow Diagrams (DFD)	Call Graphs, Source Code, System Database	instances of SSAs (e.g. ERM models, use cases)
Tool support	n/a	n/a	External tool for generating call graphs	implementation and wiki available
Degree of human involvement	domain experts define boundaries for business responsibilities	manual construction of purified DFD	no interaction needed	priorization of coupling criteria
Granularity	depends on the size of the defined business capability	most fine-grained ms candidates in terms of data operations	lowest granularity of sw based on structural and behavioural properties	n/a
Validation	demonstrated on sample domain	two case studies verified against relevant microservice principles and results of [19]	two experiemtns with complex enterprise systems (legacy vs. ms implementation)	validation via implementation and two case studies
Limitation	only conceptional approach, requires vast amount of expertise	transforming purified DFD not trivial (identifying same data operations requires expertise)	requires expressive log files to generate call graphs and identify business object relationships	generating SSAs in specified format is work intense

Table 4.3.: Comparison of Approaches, Part II

5. Preparation of Approach

This chapter provides fundamental information about the techniques and notation used in the approach. As introduced in Chapter 3, the Case Study's system specification is given in terms of detailed use cases. However, the proposed approach uses business processes as input. It is therefore necessary to provide a formal concept that assists with transforming use cases into business processes.

The following sections introduce a well known and established use case notation technique, a standardized notation to capture business processes and a structured approach to transform use case sets as business processes. Further, the extraction of data flow and control flow from business processes is discussed.

5.1. Use Cases

Use Cases are a widely adopted technique to document software system requirements. Generally, they describe the interaction between actors (usually system users) and the software system itself. In this thesis, use cases are provided as semi-structured tables (following the notation presented by Cockburn et al. [12]).

An example is given in Table 5.1: Each use case has a unique identifier and a short description, followed by necessary preconditions and a trigger that causes the execution. The standard process is the main part and describes the success steps of the Use Case. Extensions provide additional information like alternatives or exceptional processes that occur in case of an unsuccessful step.

UC 5	Show Stock Reports
Brief Description	The opportunity to generate stock-related reports is provided by the Trading System.
Precondition	The reporting GUI at the Store Client has been started.
Trigger	The Store Manager wants to see statistics about his store.
Postcondition	The report for the Store has been generated and is displayed on the reporting GUI.
Standard Process	1. The Store Manager enters the store identifier and presses the button Create Report. 2. A report including all available stock items in the store is displayed.
Extensions	(none)

Table 5.1.: Example Use Case in Tabular Form, Source: [22]

The usage of Use Cases as input for the approach has a remarkable benefit: Besides being a widely adopted technique to specify system requirements, the textual use case notation is

understandable without further technical knowledge. Neither previous knowledge in specific graphic notations like UML, nor the capability to create a complex domain model is necessary. Consequently, all sort of stakeholders (non-technical and technically experienced) are capable to provide the necessary information in terms of use cases.
However, the transformation as presented in Sec.5.3 is not always trivial and requires some manual effort to produce high quality business processes.

5.2. Business Process and Model Notation

The Business Process and Model Notation (BPMN) is a graph oriented language to describe business processes. Originally, BPMN was designed to describe activities and their control flow dependencies only [28]. Since the introduction of BPMN 2.0, it is possible to model the data needs and the data results of activities [9]. Consequently, BPMN is capable to express the control flow and to approximate the data flow of business processes [36]. In the remainder, we use BPMN instead of BPMN 2.0 for the sake of convenience.
BPMN is easy-to-use, powerful and widely adopted in academia and industry. Hence, BPMN is a suitable approach to extract the implicitly given data flow and control flow in the use case description. Sec.5.3 introduces a formal approach to generate BPMN processes from use case sets. Next, the BPMN 2.0 process definition is shortly introduced:

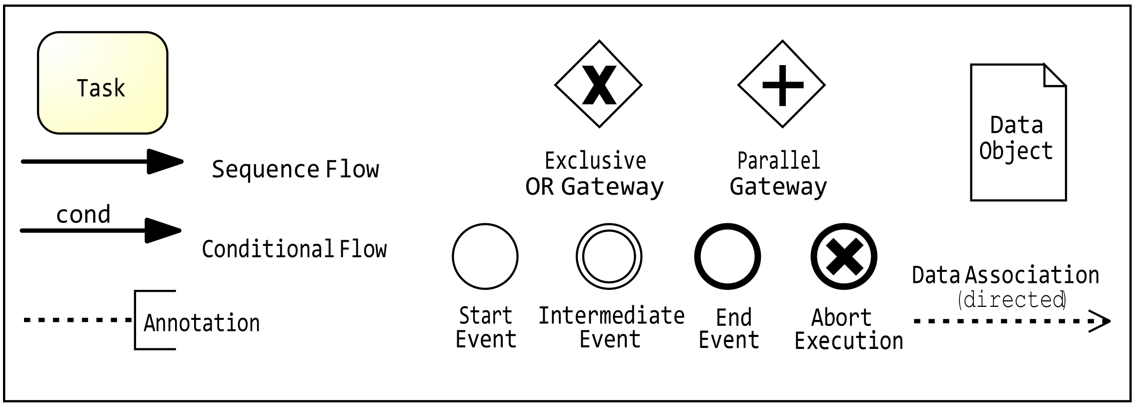

Figure 5.1.: BPMN Notation (Subset)

Fig.5.1 presents the subset of BPMN symbols, that is required for the approach presented in this thesis [1]. Control flow activities are modelled as atomic *Tasks* and connected through *Sequence Flow Arcs*. *Conditional Flow Arcs* integrate decision points into the control flow. Navigation decisions are based on the conditions related to the individual arcs. Such decision point are the *Exclusive Or Gateway* and the *Parallel Gateway*. Regarding the first one, if one of the incoming flows is triggered exactly one outgoing flow is activated based on the condition. For the latter, all outgoing flows are activated as soon as all of its incoming flows are activated. Each BPMN process starts with a *Start Event* and ends with an *End Event*. In case of several branches (due

[1] The entire specification is available at https://www.omg.org/spec/BPMN/2.0/

to Gateways), stop events need to be placed to each end. *Intermediate Events* mark any other events that occur during the process. The trigger for an event is modelled using the *Annotation* symbol. *Abort Execution Event* extend the *End Event* and marks the error-prone end of a business process.
When it comes to data, each *Task* may or may not require and/or produce data. Directed *Data Association Arcs* provide the opportunity to model data needs and data results. In case a task requires data, the corresponding *Data Objects* are connected to the task with the arrowhead attached to the task. Producing data works in the opposite direction.

5.3. Use Case Sets and BPMN Processes

To visualize the implicit data and control flow in use cases, it is necessary to transform the given use cases into BPMN models. In "Visualizing Use Case Sets as BPMN Processes" [28], Lübke et al. already elaborated an approach to visualize the control flow that is hidden in use cases. As Lübke does not use the BPMN 2.0 notation, data is not considered and hence, data flow is not part of the given approach. In the following, we will introduce a conceptional approach, which is based on [28], to transform use cases into BPMN models and extract the data flow and control flow.

5.3.1. Transform Use Case Sets in BPMN Processes

To visualize use case sets, including data flow and control flow, the following elementary steps need to be performed:

1. Create a flat use case model. Replace *include* and *extend* relationships by the associated use case

2. Generate an independent BPMN process for each use case

3. Join the generated BPMN processes based on preconditions, trigger and postconditions

4. Remove duplicate Data Objects and refactor the data associations

5. Refactor the business model i.e. divide or remove unnecessary steps, identify synonymous data objects, trigger etc.

The first step only includes simple substitutions, as the *extend* and *include* relationships simply have to be replaced by the actual use case.
Step number two represents the main part of the transformation. Fig.5.2 is an additional illustrative diagram that explains this step. First, the preconditions is assigned to the start event by using an annotation. Triggers are represented by intermediate events and as they are executed in parallel, connected by two Parallel Gateways. Each step in the use cases standard process is represented by a single task. This procedure might include some refactoring as further enlightened in the final step. Tasks that produce and/or consume data are connected to the corresponding data object. It has to be noticed, that data objects appear only once in a

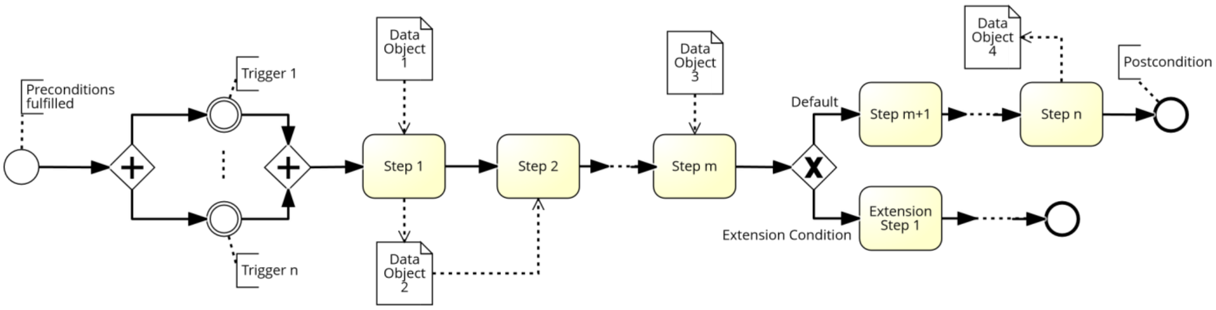

Figure 5.2.: Use Case transformed in a BPMN process

model. Given these point, it is necessary to check if a data object is already referenced by a previous task. Jumps and alternatives are modelled using Exclusive Or Gateways. Finally, the postcondition is added as annotation and connected to the end event.

Joining the use case-based BPMN models is necessary to represent the control flow and data flow within the entire system. (The flows are generally not limited to "use case borders"). For each pair of use cases (*UC A* and *UC B*), check if the postcondition of UC A exists as precondition or trigger of UC B. If this is the case, join the accompanying BPMN processes by deleting the start event (BPMN process of UC B) and the end event (BPMN process of UC B) and connect the graphs. In case that multiple use cases have the same postcondition, their BPMN processes are connected using an Exclusive Or Gateway (Fig.5.3). If several use cases have the same preconditions or triggers and therefore use the same postcondition (Fig.5.4), join them by introducing a Parallel Gateway and split the control flow. Notice, that the use cases in Fig.5.3 and Fig.5.4 are only displayed as collapsed subprocess for the sake of clarity. Second to last, the resulting data associations and objects are refactored. Duplicate Data Objects that arose from joining two processes need to be eliminated. For each set of duplicate data objects, remove all but one and reconnect the existing Data Associations to the remaining object.

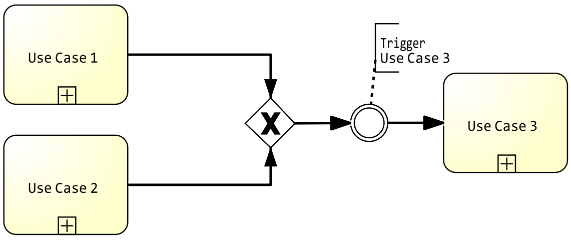

Figure 5.3.: Join BPMN processes on same postconditions

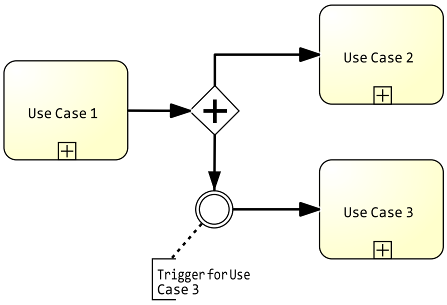

Figure 5.4.: Join BPMN processes on same preconditions or triggers

Finally, the entire business models need to be refactored. Whereas the prior steps are very algorithmic and follow a predefined concept, this step includes non-trivial work. Due to the fact that use cases are given in natural language, one has to identify synonymous data objects,steps, triggers, post- and preconditions. Moreover, a step in an use case may include two activities and has to be split up. For example, given these two steps from two different use cases:

- *The Store Manager changes the sales price of the Stock Item and commits the change by pressing enter*
- *The sale information is sent to the Inventory in order to update the Stock*

Both steps update the same Data Object, which is the Stock Item, although the second step refers to the object as *Stock*. The activity in both cases can be described as *Update inventory*. Furthermore, steps one contains two activities and needs to be split in *Change Price* and *Update Inventory*.

5.3.2. Limitations and Drawbacks of the Transformation

In Chapter 6, we present an approach that identifies microservices from the business point of view using BPMN processes as input. Nevertheless, the case study we use only provides system specification in form of detailed use cases. Therefore, the previous sections present a transformation of use cases in a set of BPMN models. However, this transformation requires a non-trivial refactoring process. In regard to data objects, it is obvious that multiple appearances of the same object (in different shaping) in a process devastate the resulting data flow. Speaking of tasks, it is necessary to examine the impact on the clustering presented in Chapter 6: It is indispensable to identify common tasks among all use cases and consequently the resulting business processes to create the structural relationship between them. If not, the tasks would be arranged in a circle-like order and the resulting clusters would correspond to the single business processes. Structural dependencies between processes like using the same functionalities would disappear.

For this reasons, the refactoring process has to be conducted conscientiously. If this is not the case or synonyms etc. are not identified, the results of the approach might not be satisfactory.

5.4. Control Flow and Data Flow in BPMN Processes

The approach uses structural and data object dependencies extracted from the control flow and data flow in order to build graphs, generate clusters and identify microservices. For this reason, the following sections explain how to extract these information from the given BPMN Processes.

5.4.1. Extract Control Flow of BPMN Processes

Extracting the Control Flow in BPMN processes is a trivial task. The BPMN was originally designed to describe the control flow in business process. All that has to be done is to delete the Data Objects and the accompanying associations. The remaining diagram visualizes the control flow, including activities and their control flow dependencies. Further information, can be extracted in various ways. For instance, counting the amount of tasks between a pair of activities provides information about their structural dependency.

5.4.2. Extract Data Flow of BPMN Processes

First of all, data flows are usually represented using specific notations of Data Flow Diagrams (DFD), i.e. a notation proposed by E.Yourdon [39]. For simplicity's sake, we relinquish to introduce another model notation and use the BPMN symbols instead.
As previously described, BPMN is only capable to express the data needs and the data results of single activities, whereas the data flow describes the flow of data in a process. By way of Fig.5.5, *Step 1* reads *Data Object 1* and writes *Data Object 2*. Despite the information about the data reads and writes of *Step 1*, it is not possible to determine without further knowledge, if any information of *Data Object 1* is used to write into *Data Object 2*. Usually, this information has to be provided by system experts.
However, the approach presented in this thesis aims to reduce the required expertise or at least the additional information that is necessary when applying the approach. As a consequence, it is fundamental to approximate the data flow based on the data needs and writes of each activity. The approximation of the data flow works similar to the previous process. First of all, control flow related parts like sequence flows arcs, gateways, events and triggers are deleted. The remaining parts are tasks, data objects and data associations. Now, the tasks are not connected to their previous neighbours, with which they might exchange data, where the data exchange is synonymous with the flow of data. To re-establish the possible data flow, follow the previously deleted control flow and reconnect the tasks with data association arcs by applying the following rules:

- Connect a pair of tasks if previously connected by a control flow arc and if another data object access happens in the course of the control flow (cf. Fig5.5)
- Replace gates by using two data association arcs (cf. Fig.5.6 and Fig.5.7).
- Remove the remaining tasks that are not connected by data association arcs (cf. Fig.5.8)

The remaining Graph contains all relevant tasks, the data objects and data associations that indicate the flow of data.

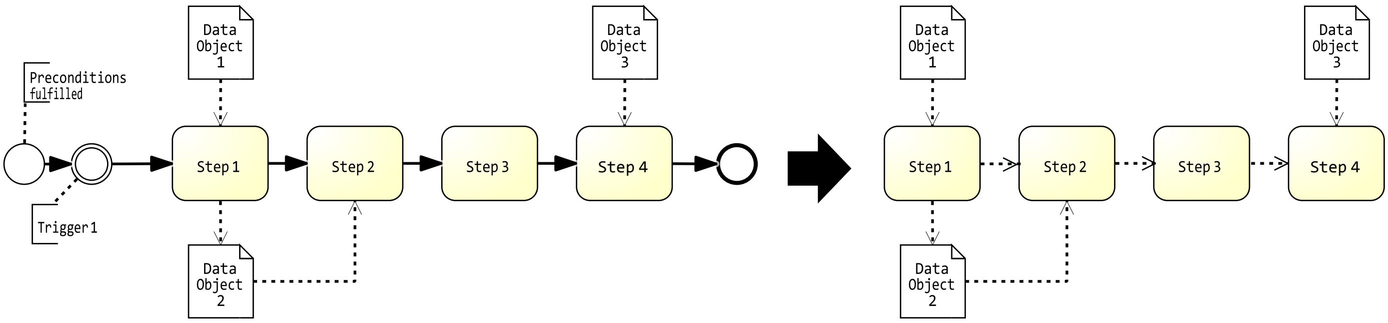

Figure 5.5.: Restore data flow connection

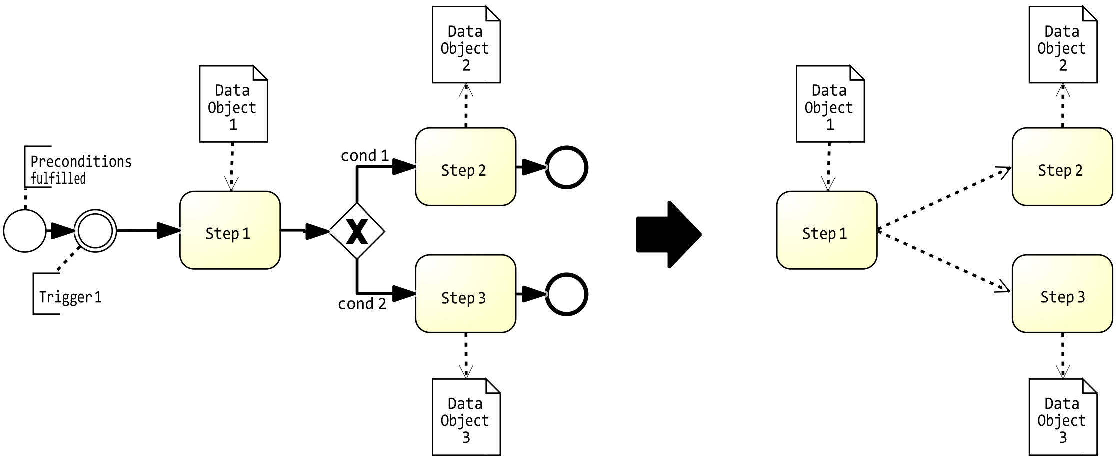

Figure 5.6.: Split data flow connection

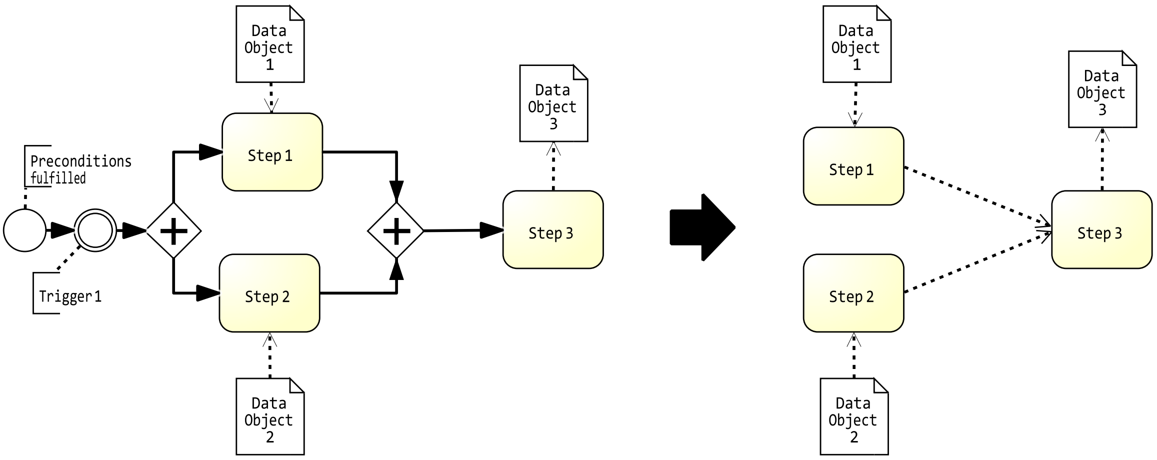

Figure 5.7.: Merge data flow connection

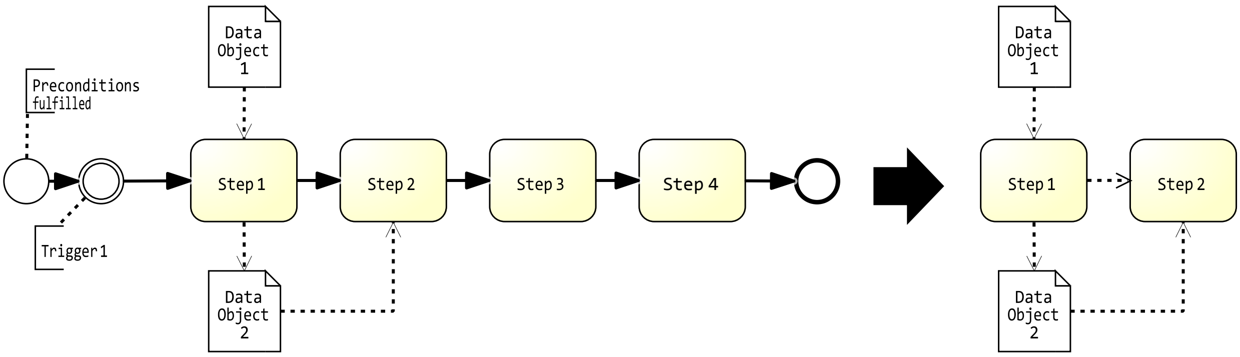

Figure 5.8.: Remove unnecessary tasks

6. Solution

This chapter presents the solution to tackle the issue of microservice identification. As noted in the *State of the Art* chapter 4, existing approaches support two initial situations: They either conduct the extraction of microservices from existing (monolithic) systems or they are based on microservice greenfield development. Both types have their advantages and disadvantages. Existing systems, for instance, provide more information about the system specification and requirements. Legacy code and log files can be used to extract data dependencies or process structures. However, shortcoming in the design of the legacy application might have an impact on the extracted information and influence the microservice extraction in a negative manner. In contrast, greenfield development is not affected by any previously committed design decisions. As a matter of fact, the greenfield approach can be applied to existing systems as well by discarding legacy code and additional information that arose during the development. Solely the system requirements that existed before the implementation started serve as input. Nevertheless, this type has to manage the identification process with less input.

The solution we propose is based on pre-existing system requirements. No existing implementation is used and consequently, the presented approach is to be classified as greenfield method.

6.1. Basic Approach

> **RQ1: Which is the most appropriate strategy to decompose a system into microservices?**

This thesis proposes a formal, graph-based microservice identification approach using clustering on control flow and data flow. It is inspired by Amiri's work on *Object-aware Identification of Microservices* [3]. In chapter 4, eight suitable approaches to identify microservices are presented and compared using well-defined criteria. Most of them require special prerequisites and cannot be applied to various types of systems, i.e. no greenfield applications, systems without meaningful VCS meta-data or the absence of log files.

By the way of contrast, Amiri proposes an approach to extract structural and data object dependencies from business point of view in order to generate possible microservice candidates. In doing so, he relinquishes to use any further information besides BPMN models. Using both, structural and data object dependencies promotes high cohesiveness and loose coupling on functional and data object level. In other words, high cohesive functionality is divided into the same microservices, together with the data objects that are accessed.

However, Sec.4.3 outlines the limitations and drawbacks of the approach. Whereas the control flow is depicted clearly, the data flow remains vague. The weight definitions regarding

data object dependencies lack formal explanation. Further, the aggregation of structural and data object dependencies, and consequently the aggregation of control flow and data flow contains a significant problem: In Amiri's approach, the aggregation is conducted by summing up two relation matrices. The matrix entries representing the dependencies highly influence the results. For instance, a large amount of data reads and writes sum up to great numbers, outweighing the structural dependencies. Thus, the identification process would be almost based on data dependencies only, discarding any identified structural dependencies.
The following sections introduces a formal approach to tackle *RQ2*. First, a basic overview of the approach is provided. Afterwards, each step is introduced in detail including alternatives and examples.

> **RQ2: What formal approach can be constructed to identify possible microservices without detailed know-how and manual effort?**

To that end, Fig.6.1 provides an overview of the proposed approach. As depicted previously, the process requires input in form of BPMN models. Therefore, specifying those models marks the beginning of the process. Afterwards, control flow and data flow need to be extracted. To avoid the ambiguity of aggregating data flow and control flow as proposed by Amiri, the approach recommends to create two independent weighted Graphs, using the information from the previous step. In the next step, a clustering algorithm determines two sets of clusters based on the weights in the graphs. At that point, the process determined a set of clusters that is based on the control flow and another one, based on the data flow. In the following, a matching process identifies commonalities between data object-based and structural-based clusters in order to create comprehensive clusters. Based on these clusters, the last step extracts microservice candidates.
RQ2 also broaches the subject of necessary know-how and the amount of manual effort. As far as that is concerned, the proposed approach does not require human interaction as soon as the BPMN models are specified. Everything beyond that is based on a structural process. Admittedly, the manual effort to conduct the process entirely is still not to be neglected, as the extraction process, graph creation and cluster matching is not yet automated. Nevertheless, the structural process enables to implement the entire approach, excluding the BPMN model specification step. However, implementing an approach is beyond the scope of the paper.

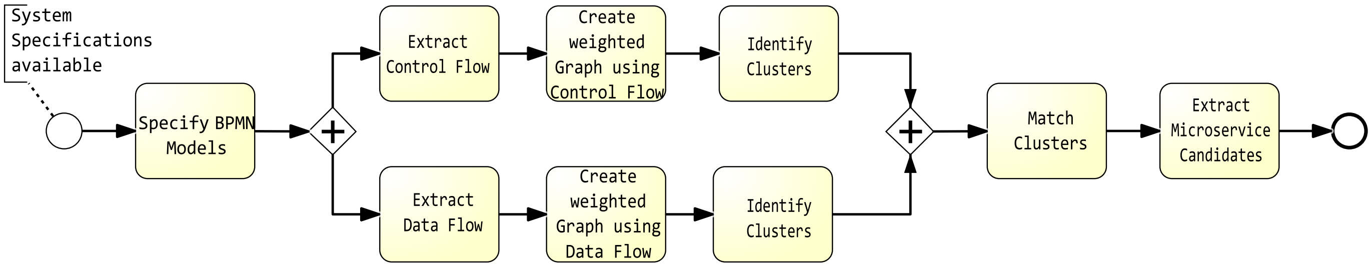
System Specifications available
Specify BPMN Models
Extract Control Flow
Create weighted Graph using Control Flow
Identify Clusters
Extract Data Flow
Create weighted Graph using Data Flow
Identify Clusters
Match Clusters
Extract Microservice Candidates

Figure 6.1.: Overview of the identification approach

6.2. Specify BPMN Models

Chapter 5, introduces the BPMN 2.0 modelling language as an easy to use, but yet powerful notation to illustrate business processes including their activities and their data needs. In the first step of the solution, those business processes need to be specified. Usually, the system specification are not directly given in form of business processes, bur rather in the form of use cases, UML models, domain models or even as textual description in natural language. Therefore, the first step consists of specifying a business model using the available system specification. This can be achieved using various approaches, for instance:

Workshops At the very beginning of a software project, technical and non-technical stakeholders can participate in a workshop to specify the business process model. As illustrated in the BPMN specification, the "primary goal of BPMN is to provide a notation that is readily understandable by all business users" [9]. Therefore, carrying out a workshop with stakeholders from various departments can produce high quality BPMN models which can be further used as input for the extraction process.

Use Cases In the case of CoCoME, the system specifications are available as use cases (cf. chapter 3). Accordingly, section 5.3 illustrates a process to transform use cases into BPMN models.

Others Business processes can be extracted in various other ways. UML Activity diagrams, for instance, are very similar to BPMN models. Van der Aalst et al. elaborated the Process Mining Manifesto [1] where he presents general techniques to extract business processes, although it is mainly event log driven. Another approach is the *BPMN Miner*, an automatic discovery tool for BPMN process models [13]. Again, the tool discovers the models dynamically, using log files of a legacy system.

6.3. Extract Control Flow

In the course of the process, activities of the business processes are clustered based on their structural dependency which is extracted from the control flow. Activities (tasks) in business processes play the role of operations in microservices, representing the functionality a service is able to offer. During the process of microservice identification, it is desired to cluster high cohesive functionality into one microservice. To achieve this, one must first extract the structural dependencies between activities in business processes.

The extraction process itself is trivial, as the business process language BPMN was designed to illustrate the control flow between activities (cf. Sec.5.2). Mainly inspired by the work of M. Amiri in *Object-aware Identification of Microservices* [3], we propose a straightforward technique to separate the control flow information from BPMN 2.0 models in section 5.4. The control flow has to be extracted for each BPMN model that was specified in the previous step.

6.4. Extract Data Flow

Besides the structural dependencies of activities, data object access plays a significant role in the definition of microservices. As depicted in the background chapter 2, microservice generally administer their own database with the data entities that belong to the bounded context of the service. Usually, data needs to be shared among services which raises the question where to place a shared data object. However, sharing data among microservices is expensive because it includes network communication instead of inter process communication. It is therefore desirable to reduce the communication between services by distributing data objects into the same microservice if they are accessed together. Like the previous section, we propose to use clustering based on data flow to identify high cohesive but loosely coupled set of data object clusters.

When BPMN 2.0 was introduced, the language was extended by the ability to represent data objects that are consumed and/or produced by the activities. Despite the fact that BPMN is still a language to illustrate the control flow of business processes, the extension provides the possibility to visualize an approximated data flow based on the data needs and writes of each activity. In Sec.5.4, we present a formal approach to extract the data flow by discarding anything but flow elements used to represent the data flow. Again, the data flow has to be extracted for each BPMN model that was specified in the previous step.

6.5. Create a weighted Graph using Control Flow

In section 6.3, the control flow is extracted from several BPMN models that represent the entire system. The visualization of the control flow information as a single graph enables to identify clusters of high cohesive functionality among all BPMN models, thus among the entire system. Therefore, we build a directed graph G whose vertices represent the tasks/activities in the BPMN models. Duplicate vertices are not allowed. Hence, activities that occur several times in different BPMN models are only represented once in the graph. The edges correspond to the control flow arcs: a pair of activities is connected if there is a direct edge in the business processes or if there is a path between them that contains only gateways. This decision is based on the heuristic, that two activities are more likely to be in the same microservice, if they are directly connected in a business process.

Speaking of the weights, we decide to assign a value of 1 to each edge notwithstanding of the nature of the connection, which is i) directly connected ii) connected via parallel gateway iii) connected via XOR gateway. Regarding the first and the second case, it is motivated by the fact that activities connected by a parallel gateway and activities that are directly connected are always executed during control flow execution. In regard of the third case, one can argue that the probability of a condition influences the weight of a connection. For instance, a task has two subsequent tasks that are connected through an exclusive OR gateway and conditional flows. One of the tasks, the "main task", is more likely to be the successor as the alternative. Hence, the edges need a different weight. However, the information regarding the probability is usually not available and specified in business processes. Further, different weighting raises the question of the value determination. With this in mind, the generalization of all types of connections (using a weight of 1 for all edges) seems to be an appropriate solution.

In the case of duplicated control flow dependencies due to several BPMN models, the weights are summed up as a pair of connected tasks that occurs in multiple models indicates a stronger cohesion. As a result, the edge in the graph that connects the tasks in questions receives a greater weight (corresponding to the number of occurrences).
Fig.6.2 shows an exemplary BPMN process whose object-related information has already been deleted (cf. Sec.6.3). As explained in this section, the control flow information given in this BPMN model is used to create a weighted graph. The corresponding graph is illustrated in Fig.6.3.

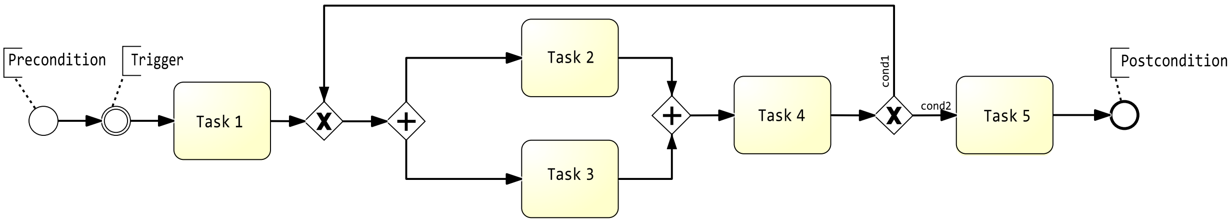

Figure 6.2.: An exemplary BPMN model illustrating the control flow only

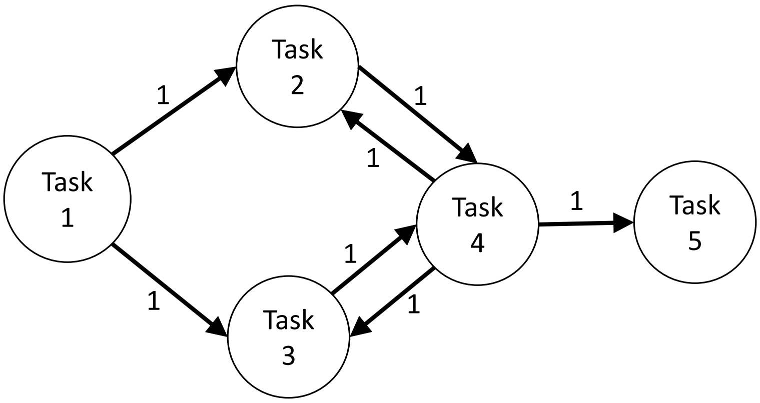

Figure 6.3.: Weighted graph using Control Flow Dependencies

6.6. Create a weighted Graph using Data Flow

To identify high cohesive data object clusters, the data flow information is visualized as a weighted graph, which is similar to the activity graph as described in the previous sections. In this case, the vertices of a Graph G represent the data objects in the BPMN models. Like the activity graph, duplicated vertices are not allowed. Each data object is only represented once in the graph. The edges illustrate the data object dependencies extracted from the data flow.

Such dependencies are: i) data objects read by the same task ii) a data object value that is used when writing to (or creating) another data object.
Speaking of the first dependency, it is obvious that data which is read by the same task is more likely to be partitioned into the same service. Otherwise, the execution of a task would always cause at least one expensive intra-service call. Therefore, two vertices that represent a pair of data objects which are read by the same task is to be connected by an edge.
The second dependency is based on a similar heuristic. There is a certain connection between two data objects, if information of one data object is used to update or create another one. Placing the information source into another microservice as the information destination would require an inevitable cross-service communication which is meant to be prevented.
To create a weighted graph based on data flow dependencies, it is necessary to take a closer look at the extracted data flow (cf. Sec.6.4). It is noticeable that it requires additional information to decide whether two data objects have one of the proposed connections. Fig.6.4 represents an exemplary data flow diagram that was extracted from a BPMN process. In the following, we discuss different possibilities to gain the data object dependencies.

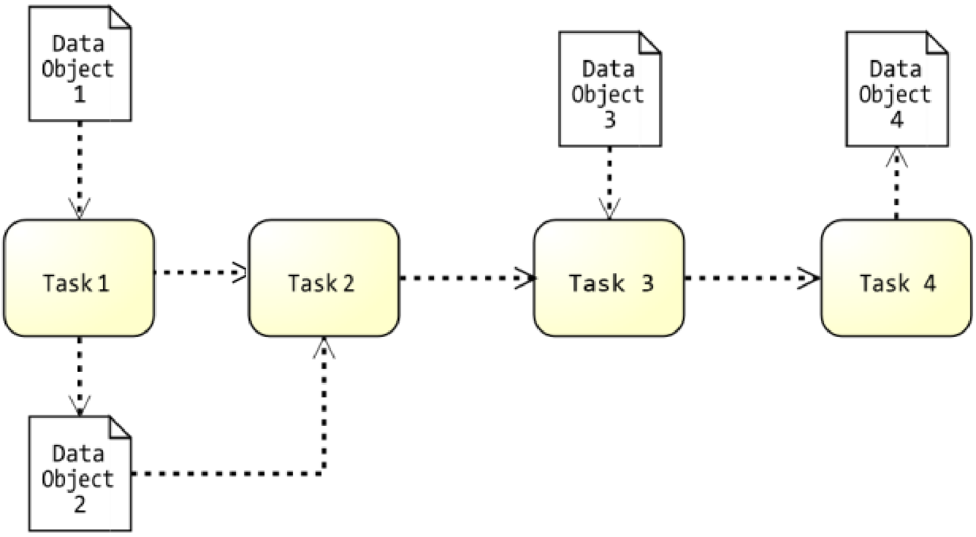

Figure 6.4.: Data Flow Diagram extracted from BPMN process

Information of one data object can flow into another one. That is the case, if a data object is updated or created using information of another data object which was read beforehand. For instance, *Task 3* processes information of *Data Object 3*, passes it to *Task 4*, which uses the information of the data object to update *Data Object 4*. Consequently, *Data Object 3 & 4* should be connected by an edge in the resulting data object graph. Yet, another possibility is that *Task 3* only reads *Data Object 3* and displays some information to the user. Further, *Task 4* only processes user input to update *Data Object 4*. Hence, *Data Object 3 & 4* are not to be connected by an edge, as there is no information flow in between.
The same line of reasoning can be applied to *Task 1*: Information of *Data Object 1* may or may not flow into *Data Object 2*, although both data accesses are executed by the same task. As a final point, the information of several data objects may flow into another one. For example, *Data Object 2*, produced by *Task 1* and *Data Object 3*, read by *Task 3*, may be used to create

Data Object 4.

Given these points, identifying data object dependencies has to be estimated. The following possibilities are available to estimate the data dependencies:

- Dependency between a pair of data objects, only if both data objects are read and written by the same task.
- Dependency between a pair of data objects, if *n* tasks[1] are in between a task that reads the first data object and another task that writes into the other data object[2].
- Use additional information to determine the actual data flow dependencies

The dependencies are expressed by connecting the vertices in question (which represent the data objects) with a weighted edge. Obviously, the third possibility is the most accurate one. The identified data object dependencies correspond to the reality. Though, one of the thesis' goals is to reduce human involvement to a minimum, so that the approach is able to run without further user interaction. Consequently, this possibility is discarded.

Regarding the first option, data object dependencies are frequently underestimated, as no information flow from one task to another is discerned. For this reason, option one is also discarded.

With this in mind, the second option seems to be the most appropriate one to determine the data flow dependencies based on the data flow graph. Still, the number *n* has to be defined. Having *n=0*, only data objects that are processed by neighbouring tasks are considered to share a dependency, therefore connected by an edge. This is reasonably similar to the control flow dependency, where only neighbouring tasks are connected by an edge as well. However, we empirically examined the data flow with *n=0* and experienced an underestimation of the existing data flow dependencies. This is due to the fact that data processing and data storing is quite often distributed among several tasks. Fig.6.5 illustrates an example:

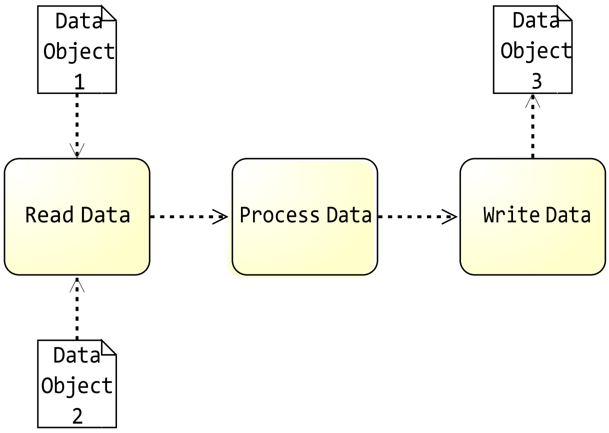

Figure 6.5.: Data Flow Diagram to demonstrate data processing and data storing

[1] n ∈ [0..], where 0 represents neighbouring tasks

[2] Following the Data Flow Arcs when counting

Two data objects are read by the first task and passed to its neighbour, which is only in charge of processing it. Finally, the merged information is stored by a third task. To represent this common pattern of data processing, a value *n>0* is required.
Nonetheless, reading and processing the data can be distributed among several tasks, depending on the granularity of the business processes. For instance, a more fine-granular business model divides the processing of *Data Object 1* and *Data Object 2* into two tasks, which still represents the same process. Thus, determining the parameter *n* highly depends on the granularity of the business processes. On the one hand, the value has to be big enough to cover data dependencies that are distributed among several tasks due to a more fine-grained process modelling. On the other hand, it should not be too big in order to prevent an overestimation of data dependencies due to data access which is executed by distant tasks. In our case, *n=1* produced the best results.
Speaking of the weights, we decided to assign a weight of 1 to each each edge notwithstanding of the connection type, which again is i) two data objects are read by the same task ii) a data object value that is used when writing to another data object. Other approaches, like the one proposed by Amiri [3], often differentiate between data reads and data writes, where the latter is generally weighted higher. However, the cross-service communication outweighs the difference between both data access types. In detail, a cross-service data read is generally much more time consuming compared to a inter-service write, due to the expensive network communication. Consequently, we propose to generalize data accesses by considering binary data dependencies only: two data objects are dependent according to the rules mentioned previously or they are not. In the case of duplicate data flow dependencies due to several tasks across various BPMN models that process the same data objects, the weights are summed up. This is motivated by the fact that multiple appearances of the same data object dependencies indicate a stronger cohesion. Fig.6.6 illustrates the Graph that is produced when applying the approach to the data flow described in Fig.6.4.

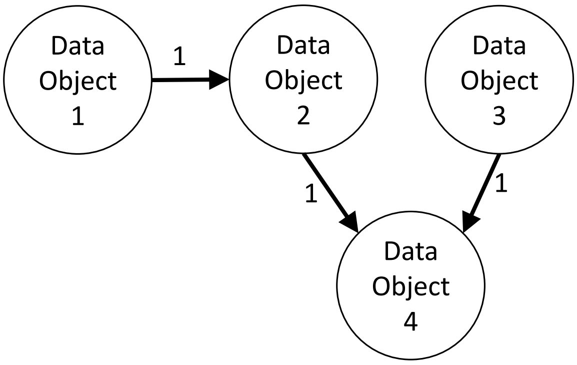

Figure 6.6.: Weighted graph using Control Flow Dependencies

6.7. Identify Cluster

The previous sections define strategies to represent the data flow and control flow dependencies as bi-directed weighted graphs. In this step of the identification process, the graphs are cut into disjunct set of nodes, called clusters. Common clustering techniques enable to identify sets of nodes with strong internal relationships and weak connections to the other clusters. The clustering is to be applied to both graphs equally, as they do not have any conceptual differences. At this point, it is important to emphasize that the elaboration of a clustering algorithm is beyond the scope of the paper. Therefore, we use existing tools for the visualization and identification of clusters.

In the course of this work, the first attempt to identify clusters involved the use of the graph visualization tool *Gephi* [3]. To layout the graph, the tool uses a force-directed algorithm based on gravity and repulsion called *Force Atlas* [6]. For the clustering, it uses a heuristic algorithm elaborated by Blondel et al. to find "high modularity partitions of large graphs" [8]. Whereas the activity clustering produced continuously constant results, the data object clustering did not. Despite using the same settings, the tool produced fair different sets of clusters when executing the algorithm. Obviously, the tool is not suitable for relatively small graphs as in the case of CoCoME.

Upon further research, we choose a tool called *Bunch*, which is a clustering tool that creates a graph decomposition by treating clustering as an optimization problem [29]. *Bunch* uses a genetic algorithm and a fitness function called *Turbo-MQ* [32]. In each iteration, the algorithm randomly picks *K* clusters and calculates *Turbo-MQ* to measure the fitness of the selected partition. In the next iteration, the algorithm tries to improve the fitness by making changes to the previous selected clusters. The algorithm stops, as soon as the overall fitness converges.

Mitchell et al. defined the "modularization quality *MQ* measurement" [32], in such a way, that it rewards intra-cluster coupling while penalizing inter-cluster coupling:

$$Turbo-MQ = \sum_{i=1}^{k} CF_i CF_i = \begin{cases} 0 & \mu_i = 0 \\ \frac{\mu_i}{\mu_i + \epsilon_i} & otherwise \end{cases}$$

The *MQ* value of a partition with *k* clusters is calculated by adding the *Cluster Factor (CF)* of each cluster. CF_i describes the normalized ratio between the total amount of internal edges μ_i and the amount of edges ϵ_i that originate in cluster i and end in another cluster. The *CF* value is between 0 (no internal edges) and 1 (no edge to another cluster), where larger values indicate a better quality of the partition.

Bunch requires the input graph in a simple textual form: The Graph is represented as list of edges, where each edge is described in a separate row by *<start node> <end node> <weight>* (without the pointed brackets). The output is in the *DOT* format [25], which is a powerful graph description language. To visualize the clustered graph, we use the open-source tool *Graphviz* [4].

[3] https://gephi.org/

[4] https://www.graphviz.org/

6.8. Match Cluster

Having both sets of clusters, it is now necessary to match the activity clusters and the data clusters in a way that reduces the required inter-microservice communication. As a first step, it is necessary to count how many times an activity cluster accesses each data object cluster. When doing this, it is not desired to differentiate between read and write accesses for the same reasons that the weighting for the object dependencies was chosen (cf. Sec. 6.6). The calculation of data access is a trivial task and only requires to examine the BPMN models one more time: For each activity that accesses a data object, identify the activity cluster the activity is located and the data object cluster the data is located. Finally, summarize for each pair of activity cluster and data cluster to obtain a relationship between the sets regarding the amount of accesses.
To process the obtained information, i.e. to match both sets of cluster, different approaches were elaborated during the course of the thesis:

- Strongest relationship matching (data object cluster oriented)
- Strongest relationship matching (activity cluster oriented)
- Clustering on data access dependency
- White box approach: Split and/or merge cluster

For the first three approaches, the respective clusters are considered from the black box point of view, as no closer look is taken into the actual clusters. Each cluster is represented as a node and connected by an undirected weighted edge, where the weights correspond to the respective amount of data accesses between the activity and the data cluster. Fig. 6.7 provides an example.

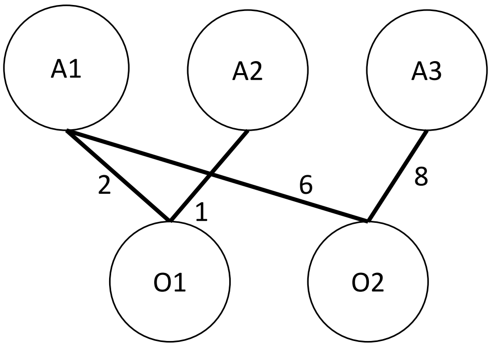

Figure 6.7.: Match Activity Cluster (A1-A3) and Object Cluster (O1-O2)

Speaking the first approach, each data object cluster is simply matched to the activity cluster that is connected by th edge with the highest weight. Further, merge object clusters that match the same activity cluster. Activity clusters that have not been matched to a data object cluster remain without accompanying data objects, i.e. not merge with another activity cluster to avoid the combination of two different bounded contexts. Whereas this approach is straight forward and easy to apply, it has drawbacks: The cluster matching from data object point of view does not consider the overall dependencies. Regarding Fig.6.7, the result would be *(O1,A1),(O2,A3),A2*. Despite the fact that *A1* accesses *O2* six times, it is outweighed by the combination *O2,A3*.

The second approach is similar as each activity cluster is matched to the object cluster that is connected by th edge with the highest weight. Activity clusters that match the same object clusters are merged. Solely object clusters that remain unmatched in the end need to be matched with one of its connected activity clusters, as the data needs to be available somewhere. The best fit in regard to data accesses is the connected activity cluster node with the highest edge weight. In this case, the result obtained from Fig.6.7 would be *(A1,A3,O2),(A2,O1)*. Like the first approach, overall dependencies are not noticed.

In order to avoid this, i.e. to consider the dependencies holistically, the third approach uses the same clustering algorithm as proposed in Sec.6.7. Thus, high cohesive activity and data object clusters are combined. In regard to the case study *CoCoME*, this approach provided satisfactory results. However, the clustering method usually merges activity clusters which can result in a too coarse-grained final microservice decomposition recommendation. So far, none of the approaches consider to split data object or activity clusters. For that, a closer look to the actual activity-data-relationships has to be taken.

In the following, we present a conceptual solution to match the two types of clusters according to their profound relationships. Hence, a white box approach is proposed: First, the clustering as presented previously is applied to achieve a first decomposition. As mentioned before, this step combines high cohesive data object clusters and activity clusters. In the next step, each combined cluster is scrutinized more precisely. In the situation that a combined cluster consists only of one activity and one data object cluster, there is nothing to do. If two (or more) activity clusters are merged and reference one data object cluster, it is necessary to take a closer look at the actual data objects they reference. In case both activity clusters reference mostly the same set of data objects in that cluster, merging is useful as distributing the activities in different services causes inevitable cross-service communication. Splitting object clusters is reasonable if it becomes apparent that one part of the data objects are referenced mostly by one activity cluster and the other part by the other activity clusters. Consequently, the previously identified set of activity clusters and the data cluster is divided into smaller parts while preserving the internal cohesion between data and activities and while keeping the combined cluster small.

This solution is not yet mature and only presented conceptional. Neither a concise definition is provided nor has it been tested on several case studies. However, we believe in the potential of the solution and propose a more detailed case study in the area of cluster matching. Although we are aware that the third solution may produce too coarsely granular results, we chose this one,for now, to match the identified data object clusters and activity clusters.

6.9. Extract Microservice Candidates

So far, BPMN models were used to extract the control flow and the data flow from a system's business processes. Based on heuristics, we determined dependencies between the activities and between the data objects and visualized them as two graphs to identify clusters of dense relationships that are weakly connected to other clusters. In the previous section, we proposed different approaches to match the activity clusters and the object clusters in order to obtain combined cluster. Those clusters correspond to the microservice candidates: The activities describe the functionality that the microservice provides. The data object clusters describe the data object which the microservice has to administer. This includes the availability of interfaces to share data with other services if necessary. Those combined clusters are good candidates to become a microservice, because:

- Most of the data objects are accessed by activities within the service, which satisfies the low coupling criteria.
- Cohesive functionality is placed in the same service, which satisfies the high cohesion criteria.
- The approach reduces inter-service communication to a minimum, which enhances the performance.

This step finalizes the microservice identification approach. In the following, it is applied to the case study CoCoME (cf. Chapter 3).

7. Solution Application

This chapter applies the previously presented approach to identify microservices from the business point of view, using clustering on control flow and data flow. It is applied to the case study CoCoME, whose system specifications are defined in chapter 3. Speaking of the order, this chapter follows the process overview illustrated by Fig.6.1.

7.1. Use Cases as BPMN Models

CoCoME's system specifications are given in terms of use cases. A short overview is available in chapter 3, whereas a more detailed version can be found in the *Technical Report* [21]. We decide to omit *UC 8 - Product Exchange* as independent BPMN model, because both reference sets either did not take it into consideration or implemented it differently. However, it was added as extension to *UC 1* as single activity named *Product Exchange*. In the same way, *UC 2 - Manage Express Checkout* is added to *UC 1* as single activity named *Manage Express Checkout*. *UC 2* extends *UC 1* and therefore, has to be associated with *UC 1* anyway. Fig.7.1 illustrates *UC 1,2* and *8* as BPMN model. The remaining BPMN models are available in the appendix (cf. A.1). For the sake of clarity, the models are not yet joined as described in section 5.3. Apart from Fig.7.1, each use case is illustrated as single BPMN process. However, the BPMN models that represent *UC 3* and *UC 4* as well as *UC 4* and *UC 1* need to be joined, as preconditions and postconditions are equal.

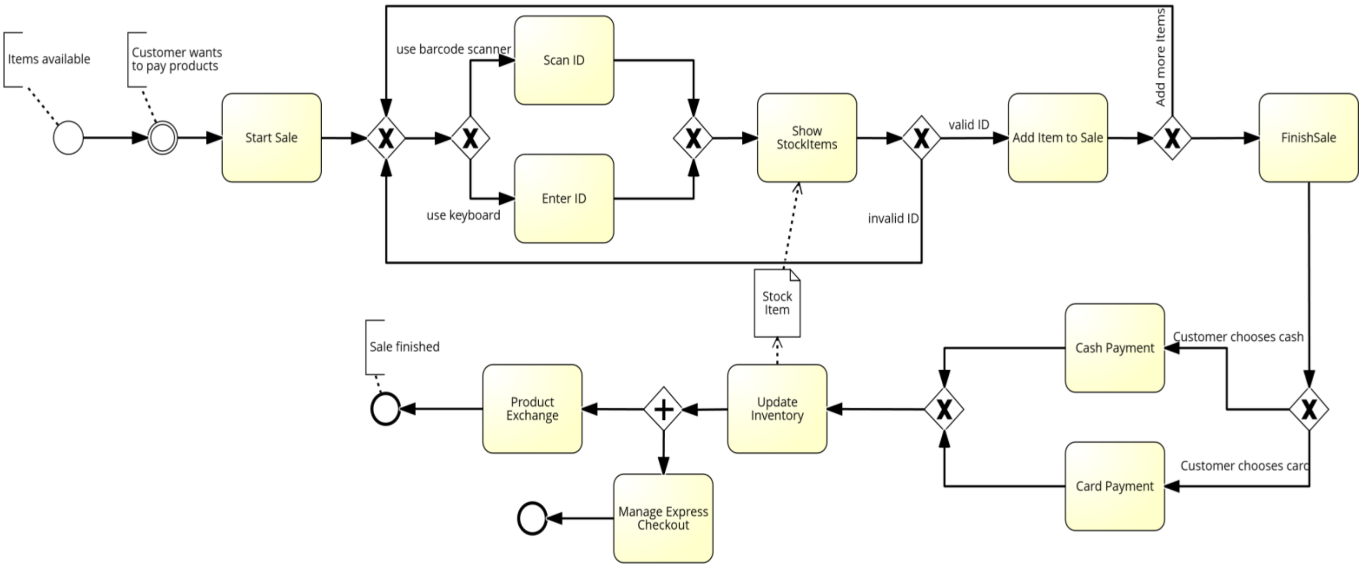

Figure 7.1.: UC1-Process Sale (with UC2 and UC8)

7.2. Extracted Control Flow

In this step, the control flow is extracted from the BPMN models. Sec. 6.3 provides the detailed extraction process, but in a word, everything but the control flow elements are deleted. Fig.7.2 presents the control flow for *UC 3*. The remaining control flow diagrams are available in the appendix (cf. A.2).

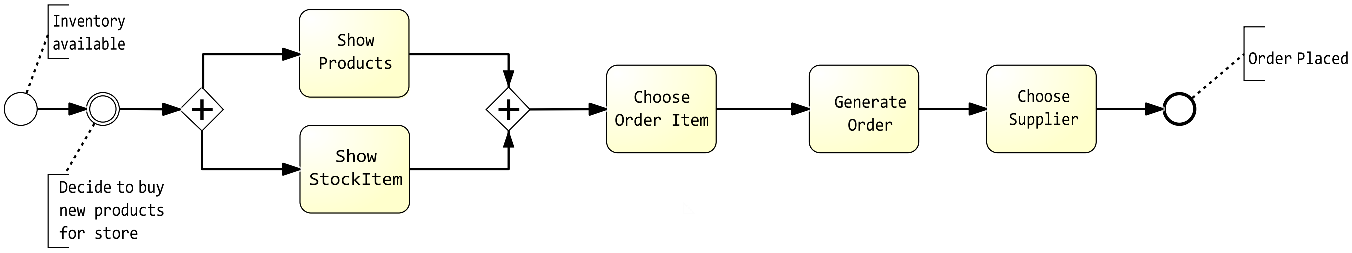

Figure 7.2.: Control Flow UC3 - Order Products

7.3. Extracted Data Flow

Like the previous step, the data flow is extracted from the BPMN models as described in Sec.6.4. Fig.7.2 presents the data flow for *UC 3*. The remaining data flow diagrams are available in the appendix (cf. A.3).

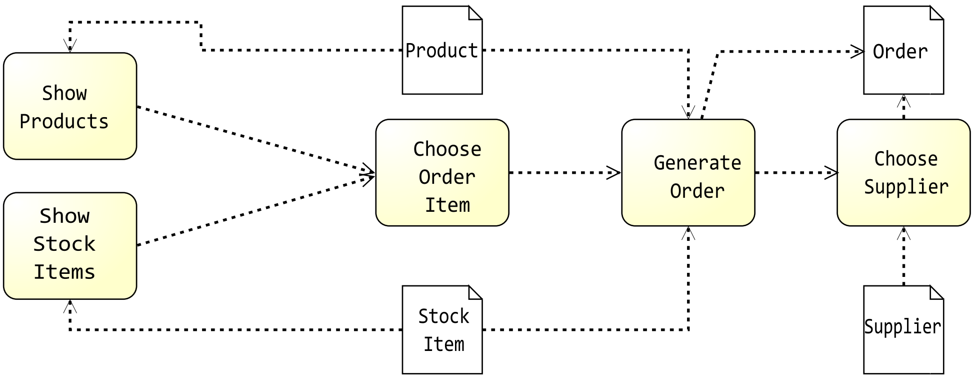

Figure 7.3.: Control Flow UC3 - Order Products

7.4. Control Flow Graph

Fig.7.4 illustrates the control flow graph of CoCoME which is derived from the control flow diagrams.

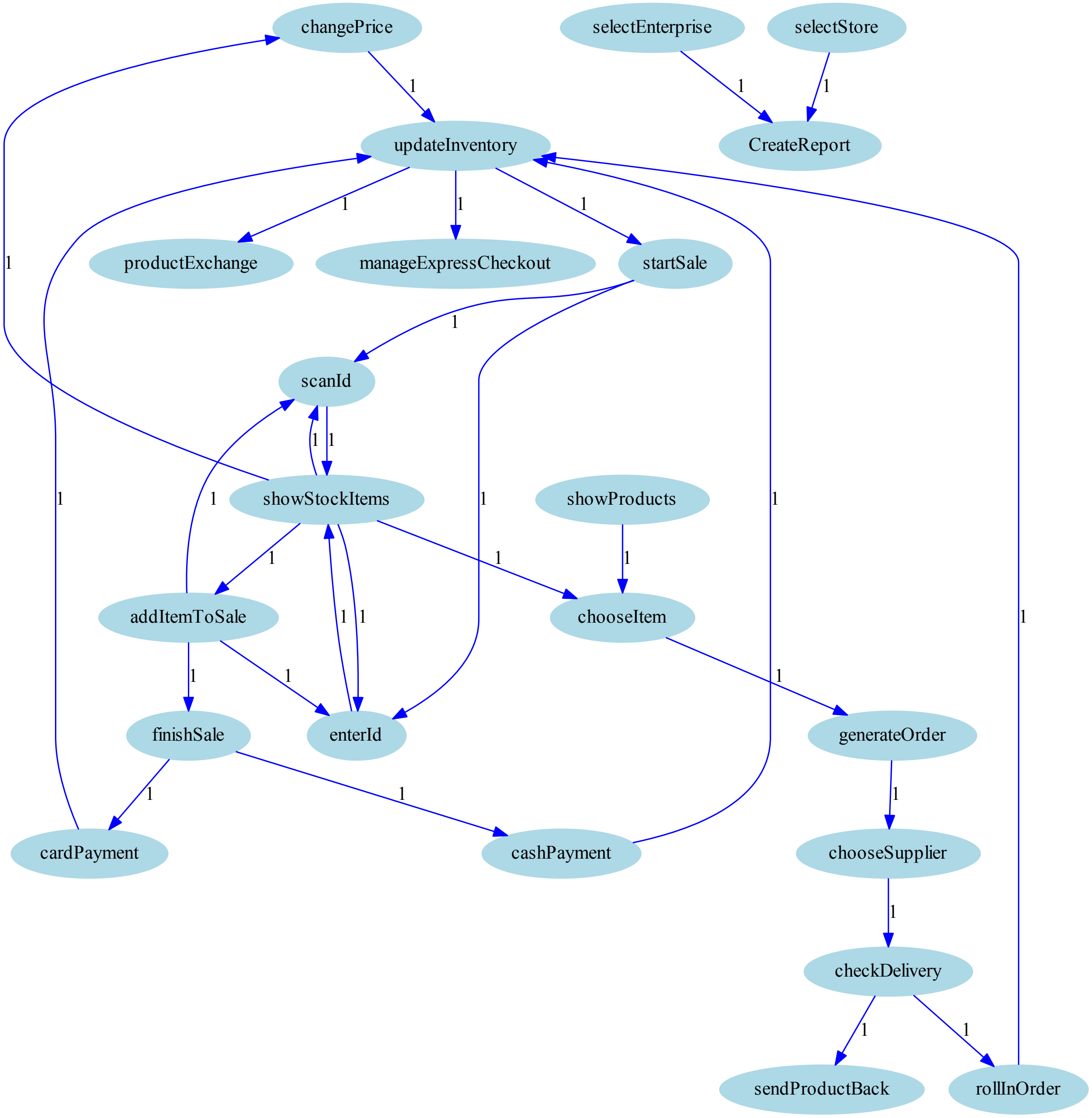

Figure 7.4.: Control Flow Information as Graph CoCoME

7.5. Data Flow Graph

Fig.7.5 illustrates the data flow graph of CoCoME which is derived from the data flow diagrams. When identifying the data flow dependencies, it is necessary to choose a value for the parameter *n*, that describes the maximum distance between a pair of activities that consume and produce a data object. We use *n=1*, as it produces the most appropriate results compared to the final microservice decomposition.

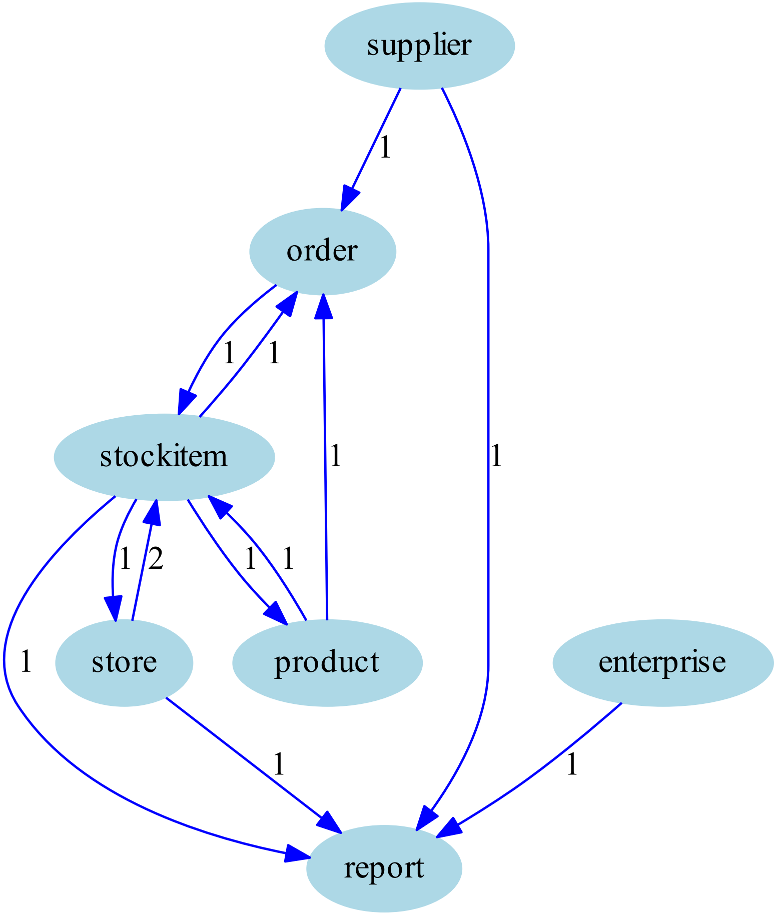

Figure 7.5.: Data Flow Information as Graph CoCoME

7.6. Activity Clusters

Fig.7.6 shows the activity clusters that are identified using the clustering algorithm described in Sec.6.7.

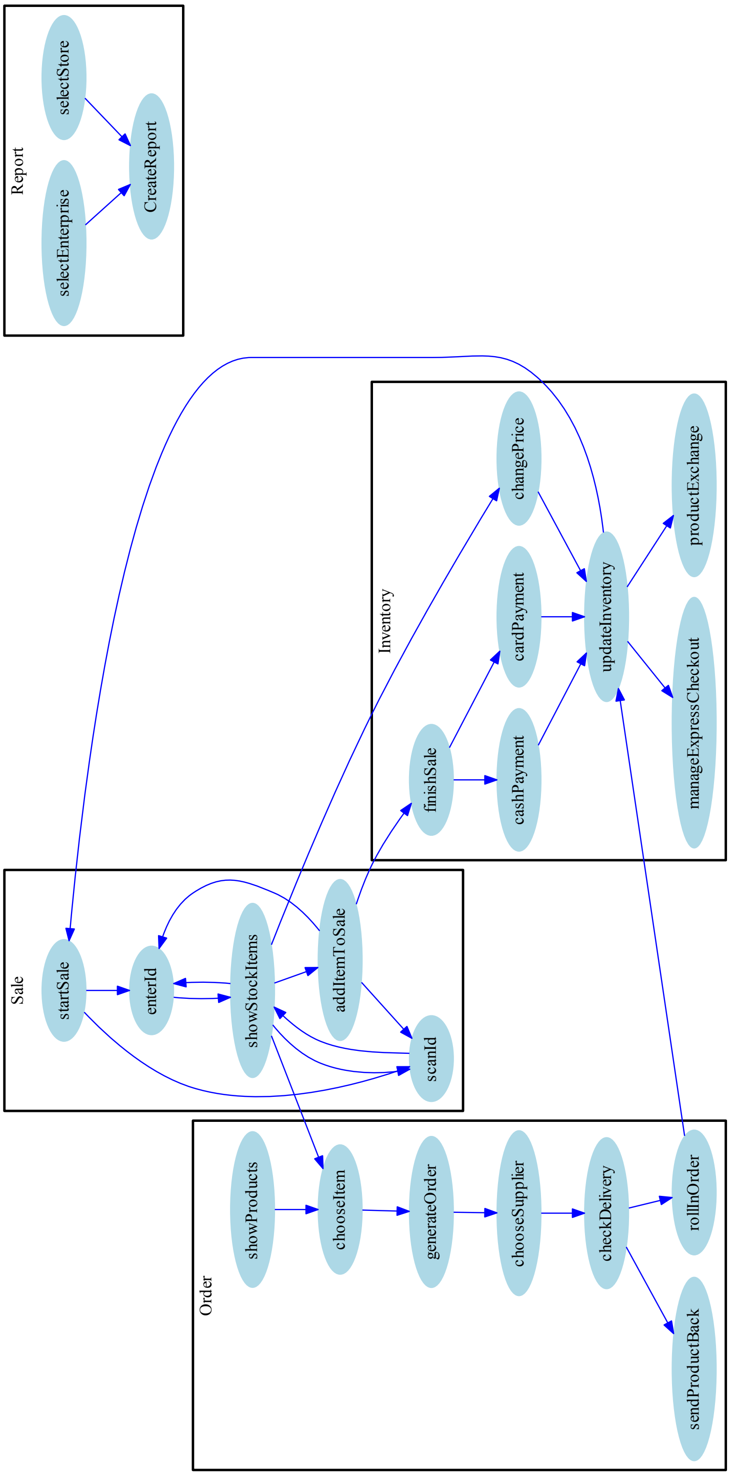

Figure 7.6.: Clustering on CoCoME's Activities

7.7. Data Object Clusters

Fig.7.6 shows the data object clusters that are identified using the clustering algorithm described in Sec.6.7.

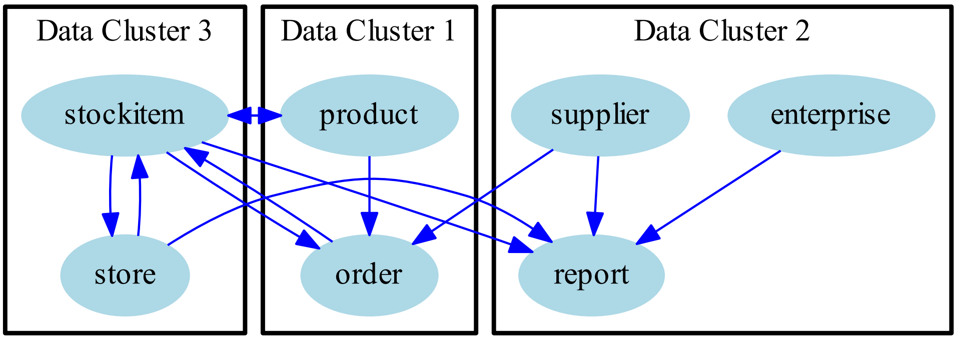

Figure 7.7.: Clustering on CoCoME's Data Objects

7.8. Match Clusters

7.9. Extract Microservice Candidates

8. Evaluation

In the introduction chapter, the *Research Questions* are presented. The third question is about the elaborated approach and its evaluation.

> **RQ3: What is the accuracy of the approach?**

To tackle this question, a *Goal Quality Metrics Plan* (GQM) is introduced to specify the key aspects of the evaluation. In a word, the elaborated approach is used to identify a set of microservice candidates which is compared to two reference sets of microservices.
In the following, a *GQM Plan* is introduced to specify what exactly needs to be evaluated. Also, metrics to measure the results of the comparison are introduced. Second to last, the two reference sets are illustrated before the actual results of the approach are finally depicted.

8.1. GQM Plan

Basili et al. originally proposed the *GQM Plan* (Goal Quality and Metrics) as a paradigm in software engineering but it is further extended to other engineering disciplines [5]. Based on a precise and structured procedure, the paradigm enables a high traceability and assessment of the engineering process. The main purpose is to specify the goals for a project, illustrate the data to define these goals and provide an environment to interpret the collected data. In the case of this thesis, the *GQM Plan* is used to clarify the desired intention of the evaluation in order to prevent unnecessary metrics and measurements and consequently reduce the expenditure of work.
The *GQM Plan* is a Top-Down approach and divided in three fundamental steps that precede the measurement and evaluation of results. First, the goal of the evaluation is defined on a conceptual level. Seconds, questions are delineated to achieve the specific goal. Finally, to answer the questions in a measurable way, metrics have to be defined that are associated with the questions.
In the following, the *GQM Plan* for this the consecutive evaluation is illustrated:

- **G1:** Determine the accuracy of the approach
- **G1.Q1:** What is the *Precision and Recall* of the identified microservices compared to the reference amount?
- **G1.Q1.M1:** Precision and Recall
- WAS WAR HIER NOHCMAL MIT DEN ZYKLISCHEN ABHÄNGIGKEITEN

8.2. Metrics

Using metrics is mandatory to measure the quality of the elaborated approach. In this case, it is required to choose a metric to classify a set of instances, namely microservices, regarding their relevance. Two reference sets are available as further depicted in Sec.8.3.
A metric that is capable to measure the relevance of a set of instances compared to a reference set is *Precision and Recall.* In subsequent, the proposed metric is briefly presented.

8.2.1. Precision and Recall

Precision and Recall is a classification metric that measures the relevance of retrievable items with respect to a reference set [10]. Commonly, two distinctions for items in the reference set are made: First, Retrieved or not Retrieved. More precisely, an item is retrieved if it is part of the selected items and vice versa. Secondly, Relevant or Not Relevant. As a result, all retrievable items belong to one and only one of four cells in the following matrix:

	Relevant	Not Relevant	Sum
Retrieved	$N_{ret\cap rel}$	$N_{ret\cap\overline{rel}}$	N_{ret}
Not Retrieved	$N_{\overline{ret}\cap rel}$	$N_{\overline{ret}\cap\overline{rel}}$	$N_{\overline{ret}}$
Sum	N_{rel}	$N_{\overline{rel}}$	N_{total}

Table 8.1.: Retrieval Matrix, Source: [10]

Recall describes the completeness of the retrieval. In other words, how many relevant items are selected in regard to all possible relevant items.

$$Recall = \frac{N_{ret\cap rel}}{N_{rel}}$$

Precision illustrates the purity of the retrieval because it puts into proportion the number of retrieved relevant items and the number of all retrieved items.

$$Precision = \frac{N_{ret\cap rel}}{N_{ret}}$$

It is important to notice that $N_{\overline{ret}}$ and $N_{\overline{rel}}$ are not part of the formulas. With that in mind, it is possible to apply *Precision and Recall* to the prevalent evaluation scenario. With respect to Table 8.1, the reference set that is used forms the relevant items, or N_{rel}. Accordingly, the set of microservices identified by the proposed approach constitutes the retrieved items, or N_{ret}. The remaining part which are the non-relevant and non-retrieved items ($N_{\overline{ret}\cap\overline{rel}}$) are to be unimportant. The following list draws the analogy between Table 8.1 and the predominant evaluation scenario:

- **True Positives:** $N_{ret \cap rel}$, identified microservices that have a similar partner in the reference set
- **False Positives:** $N_{ret \cap \overline{rel}}$, identified microservices that do not have a similar partner in the reference set
- **False Negatives**: $N_{\overline{ret} \cap rel}$, microservices in the reference set that have not been discovered by the proposed approach
- **True Negatives**: $N_{\overline{ret} \cap \overline{rel}}$, microservices that are neither discovered by the approach, nor part of the reference set [1]

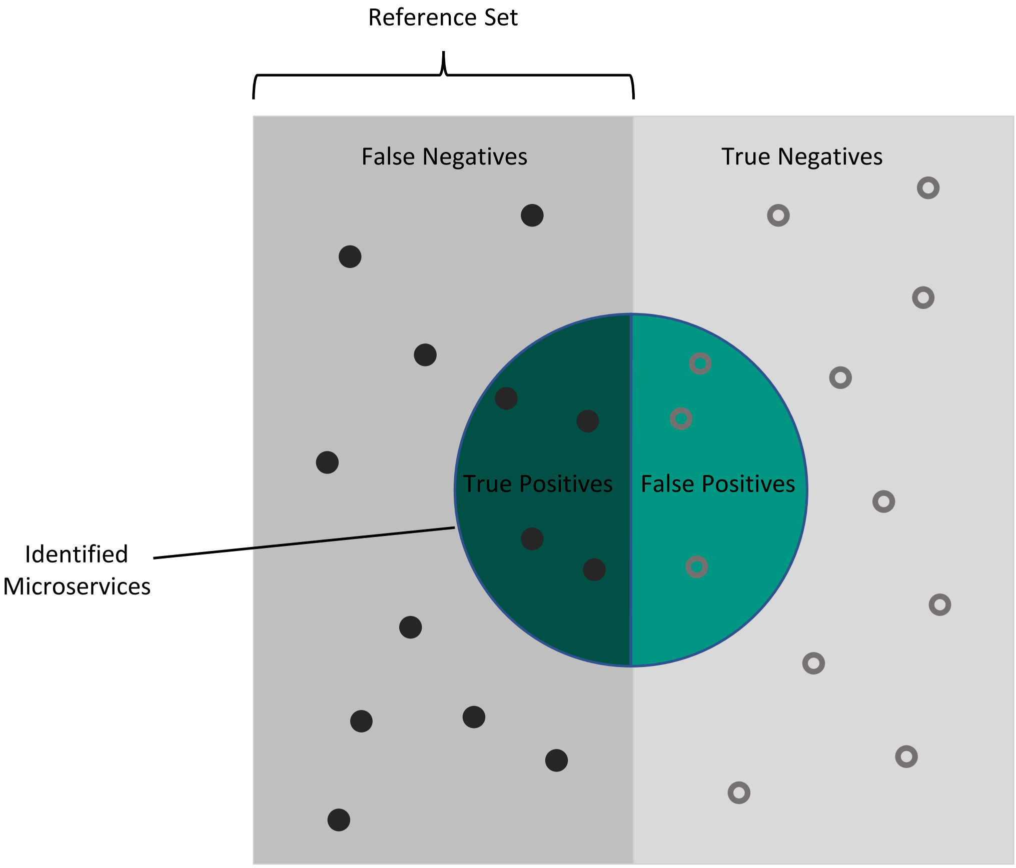

Figure 8.1.: Precision and Recall for Microservices

[1] Note, that this amount consists of all imaginable microservices and is therefore an infinite set. As it is not used to calculate either of the metrics, it is negligible.

8.2.2. Some more Metrics if necessary

//Hier noch erklären

8.3. Reference Sets

To evaluate the approach, the identified set of microservices (cf. Sec.8.4) is compared to two alternative decompositions of the case study: First, a decomposition proposed in the paper *Identifying Microservices Using Functional Decomposition* [37] and second, a set of microservices which we manually identified.

8.3.1. Reference Set 1: Functional Decomposition Approach

Identifying Microservices Using Functional Decomposition [37] is a systematic approach to find a appropriate partition of a system into microservices. This paper emerged as a result of the collaboration of the Academic College Tel-Aviv Yafo, the Karlsruhe Institute of Technology and the Southwest University China and uses CoCoME as demonstrator as well.
As depicted in Sec.4.2, Tyszberowicz et al. utilize the Use Case specifications of CoCoME [22] as input for their decomposition approach. Several external tools are used to extract verbs an nouns from the use cases that serve as *system operations* and *state variables*. Irrelevant noun, verbs and synonyms are eliminated via brainstorming. The relationships between the aforementioned concepts are stored in a relation table. A relation exists, if a *system operation* reads or updates a *state variable*. Thereupon, the relation table is visualized as a weighted graph, which enables to identify clusters of dense relationships. Each cluster serves as a microservice candidate.
As mentioned in Sec.4.3, the compulsory and non-trivial revision of nouns and verbs to eliminate synonyms etc. is a substantial disadvantage. However, the evaluation results in Tyszberowicz's approach demonstrate, that the identified microservices are good candidates for a microservice-based system decomposition of CoCoME. The aforementioned evaluation includes a comparison to three independent software projects that implemented CoCoME. Two groups identified, apart from the naming, the same set of microservices. The third group identified a more detailed decomposition of the case study, but a revision reveals that the additional microservices are only a refinement of the proposed microservices.
The following microservices are identified:

- *Reporting:* TODO Features auflisten
- *StockOrder:*
- *Sale:*
- *ProductList:*

8.3.2. Reference Set 2: Manual Decomposition

In the course of this thesis, we implemented a microservice-based version of CoCoME. The microservice identification process itself was terminated before the literature review for the thesis started. Moreover, we were not aware of the microservice decomposition proposed by Tyszberowicz et al. [37] by the time we identified possible microservice candidates. Consequently, the process was conducted without bias.

The identification process itself was conducted manually and supported by the previous knowledge of the CoCoME domain. Beside the use case specification, we used a monolithic implementation of CoCoME, the Hybrid Cloud-based Variant [21], as information resource to discover requirements, functionality, dependencies and finally decompose the system into loosely coupled and high cohesive microservices.

Once more, the time consuming and difficult discovery process clarified the necessity of a structured and formal approach to identify microservices.

The following four microservices were identified:

- *Stores- and Sales service*
- *Product*
- *Order*
- *Reports*

8.3.3. Differences between both sets

Roughly same services but differences in functionality

8.4. Results

8.4.1. Identified Microservices

//Hier veranschaulichen was unser Ansatz gefunden hat

9. Discussion

9.1. Using Use Cases as Input

- Das Transformieren mit Data Needs ist nicht komplett trivial –> Falls aber BPMN als Input bereits vorliegen, wäre es perfekt

9.2. Control Flow Graph...

- Bei direkt verbundenen ist es eindeutig. Bei Parallel Gateway auch, da beide gleich oft ausgeführt werden. Problematisch wird es bei XOR, da man nicht weiß, welche der wege mit welcher Wahrscheinlichkeit genommen werden

Rein das wir kein extra DFD wollen als inpout sonder einfachen input verarbeiten –> mehr testen und eventuell doch den DFD extra extrahieren

10. Conclusion

10.1. Outcomes

10.2. Limitations and Future Work

Granularität der Business Prozesse...

Matching Algo weiterführen

Bibliography

[1] Wil van der Aalst et al. “Process Mining Manifesto”. In: *Business Process Management Workshops*. Ed. by Florian Daniel, Kamel Barkaoui, and Schahram Dustdar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 169–194. ISBN: 978-3-642-28108-2.

[2] N. Alshuqayran, N. Ali, and R. Evans. “A Systematic Mapping Study in Microservice Architecture”. In: (Nov. 2016), pp. 44–51.

[3] M. J. Amiri. “Object-Aware Identification of Microservices”. In: (July 2018), pp. 253–256. ISSN: 2474-2473. DOI: 10.1109/SCC.2018.00042.

[4] Luciano Baresi, Martin Garriga, and Alan De Renzis. “Microservices Identification Through Interface Analysis”. In: (2017). Ed. by Flavio De Paoli, Stefan Schulte, and Einar Broch Johnsen, pp. 19–33.

[5] Victor R. Basili. *Software Modeling and Measurement: The Goal/Question/Metric Paradigm*. Tech. rep. College Park, MD, USA, 1992.

[6] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. “Gephi: an open source software for exploring and manipulating networks”. In: *Third international AAAI conference on weblogs and social media*. 2009.

[7] Niko Benkler. *From Traditional Development to Continuous Deployment: Strategies and Practices in CI/CD Pipelines*. Accessed on 20.01.2019. URL: https://github.com/Benkler/Proseminar/blob/master/Niko_Benkler_Proseminar.pdf.

[8] Vincent Blondel et al. “Fast Unfolding of Communities in Large Networks”. In: *Journal of Statistical Mechanics Theory and Experiment* 2008 (Apr. 2008). DOI: 10.1088/1742-5468/2008/10/P10008.

[9] *BPMN OMG*. Accessed on 25.02.2019. URL: https://www.omg.org/spec/BPMN/2.0/.

[10] Michael Buckland and Fredric Gey. “The relationship between recall and precision”. In: *Journal of the American society for information science* 45.1 (1994), pp. 12–19.

[11] R. Chen, S. Li, and Z. Li. “From Monolith to Microservices: A Dataflow-Driven Approach”. In: (Dec. 2017), pp. 466–475. DOI: 10.1109/APSEC.2017.53.

[12] Alistair Cockburn. *Writing Effective Use Cases*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000. ISBN: 0201702258.

[13] Raffaele Conforti et al. “BPMN Miner: Automated discovery of BPMN process models with hierarchical structure”. In: *Information Systems* 56 (2016), pp. 284–303. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2015.07.004. URL: http://www.sciencedirect.com/science/article/pii/S0306437915001325.

[14] Adambarage Anuruddha Chathuranga De Alwis et al. "Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems". In: (2018). Ed. by Claus Pahl et al., pp. 37–53.

[15] D. Escobar et al. "Towards the understanding and evolution of monolithic applications as microservices". In: (Oct. 2016), pp. 1–11.

[16] Lewis Fowler. *Microservices.* Accessed on 17.01.2019. URL: `https://martinfowler.com/articles/microservices.html`.

[17] P. Di Francesco, P. Lago, and I. Malavolta. "Migrating Towards Microservice Architectures: An Industrial Survey". In: (Apr. 2018), pp. 29–2909.

[18] Jonas Fritzsch et al. "From Monolith to Microservices: A Classification of Refactoring Approaches". In: *CoRR* abs/1807.10059 (2018). arXiv: `1807.10059`. URL: `http://arxiv.org/abs/1807.10059`.

[19] Michael Gysel et al. "Service Cutter: A Systematic Approach to Service Decomposition". In: (2016). Ed. by Marco Aiello et al., pp. 185–200.

[20] S. Hassan, N. Ali, and R. Bahsoon. "Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity". In: (Apr. 2017), pp. 1–10.

[21] Robert Heinrich, Kiana Rostami, and Ralf Reussner. "The CoCoME Platform for Collaborative Empirical Research on Information System Evolution". In: (Jan. 2016). DOI: `10.5445/IR/1000052688`.

[22] Sebastian Herold et al. "CoCoME - The Common Component Modeling Example". In: *The Common Component Modeling Example: Comparing Software Component Models.* Ed. by Andreas Rausch et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 16–53. ISBN: 978-3-540-85289-6. DOI: `10.1007/978-3-540-85289-6_3`. URL: `https://doi.org/10.1007/978-3-540-85289-6_3`.

[23] G. Kecskemeti, A. C. Marosi, and A. Kertesz. "The ENTICE approach to decompose monolithic services into microservices". In: (July 2016), pp. 591–596.

[24] S. Klock et al. "Workload-Based Clustering of Coherent Feature Sets in Microservice Architectures". In: (Apr. 2017), pp. 11–20.

[25] Eleftherios Koutsofios, Stephen North, et al. *Drawing graphs with dot.* Tech. rep. Technical Report 910904-59113-08TM, AT&T Bell Laboratories, Murray Hill, NJ, 1991.

[26] Alessandra Levcovitz, Ricardo Terra, and Marco Tulio Valente. "Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems". In: *CoRR* abs/1605.03175 (2016). arXiv: `1605.03175`. URL: `http://arxiv.org/abs/1605.03175`.

[27] J. Lin, L. C. Lin, and S. Huang. "Migrating web applications to clouds with microservice architectures". In: (May 2016), pp. 1–4.

[28] D. Lubke, K. Schneider, and M. Weidlich. "Visualizing Use Case Sets as BPMN Processes". In: *2008 Requirements Engineering Visualization.* Sept. 2008, pp. 21–25. DOI: `10.1109/REV.2008.8`.

[29] Spiros Mancoridis et al. "Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures". In: (Apr. 1999).

[30] G. Mazlami, J. Cito, and P. Leitner. "Extraction of Microservices from Monolithic Software Architectures". In: (June 2017), pp. 524–531.

[31] Genc Mazlami. *Algorithmic Extraction of Microservices from Monolithic Code Bases.* Accessed on 20.01.2019. URL: `https://www.merlin.uzh.ch/contributionDocument/download/10978`.

[32] B. Mitchell, M. Traverso, and S. Mancoridis. "An architecture for distributing the computation of software clustering algorithms". In: *Proceedings Working IEEE/IFIP Conference on Software Architecture.* Aug. 2001, pp. 181–190. DOI: `10.1109/WICSA.2001.948427`.

[33] I. J. Munezero et al. "Partitioning Microservices: A Domain Engineering Approach". In: (May 2018), pp. 43–49.

[34] Chris Richardson. *Microservices: Decomposing Applications for Deployability and Scalability.* Accessed on 08.01.2019. May 2014. URL: `https://www.infoq.com/articles/microservices-intro`.

[35] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices". In: *IEEE Access* 5 (2017), pp. 3909–3943.

[36] Silvia von Stackelberg et al. "Detecting Data-Flow Errors in BPMN 2.0". In: *OJIS* 1 (2014), pp. 1–19.

[37] Shmuel Tyszberowicz et al. "Identifying Microservices Using Functional Decomposition". In: (2018). Ed. by Xinyu Feng, Markus Müller-Olm, and Zijiang Yang, pp. 50–65.

[38] Zhiping Luo UU, Michel Korpershoek, and AnaMaria Oprescu VU. "Towards a MicroServices Architecture for Clouds". In: ().

[39] Edward Yourdon. "Structured Programming and Structured Design As Art Forms". In: *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition.* AFIPS '75. Anaheim, California: ACM, 1975, pp. 277–277. DOI: `10.1145/1499949.1499997`. URL: `http://doi.acm.org/10.1145/1499949.1499997`.

A. Appendix

A.1. BPMN Models

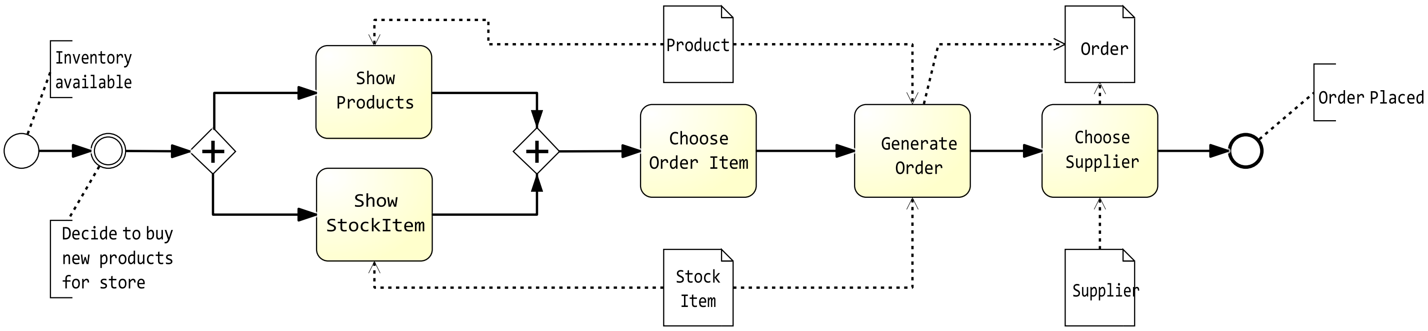

Figure A.1.: UC3 - Order Products

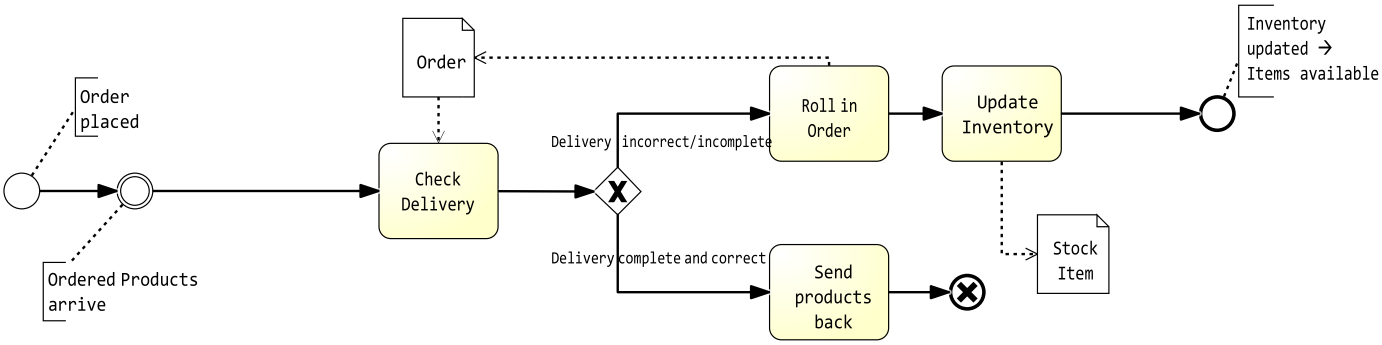

Figure A.2.: UC4 - Receive Ordered Products

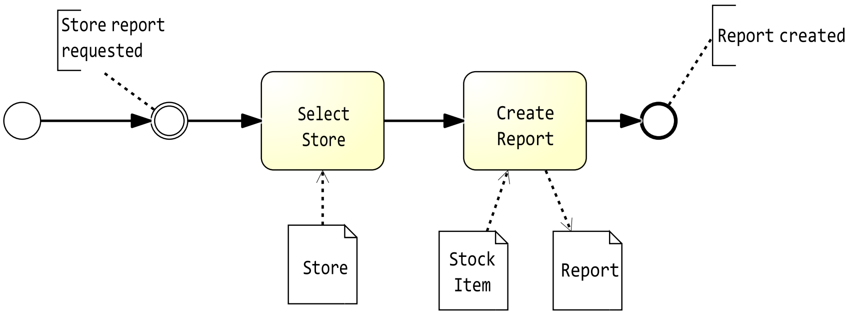

Figure A.3.: UC5 - Show Stock Reports

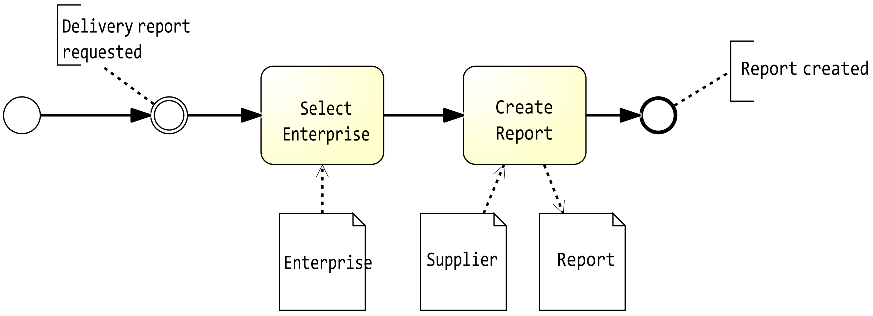

Figure A.4.: UC6 - Show Delivery Reports

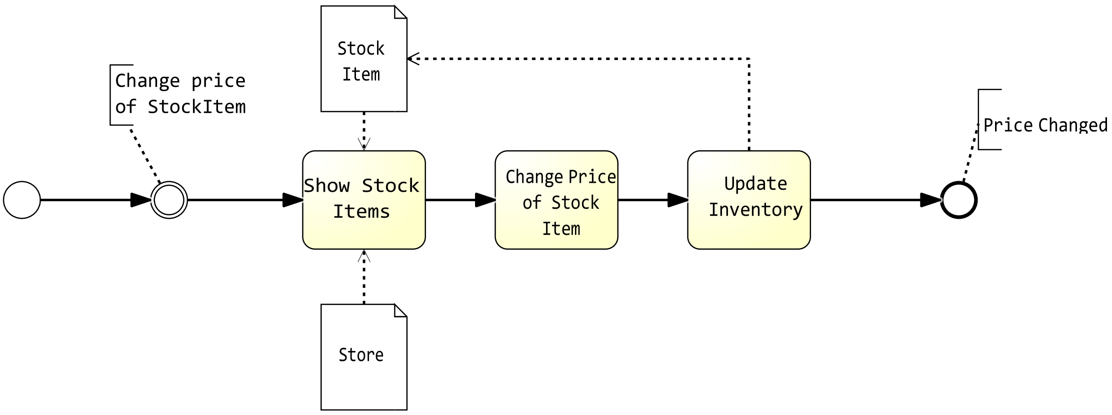

Figure A.5.: UC7 - Change Price

A.2. Control Flow Diagrams

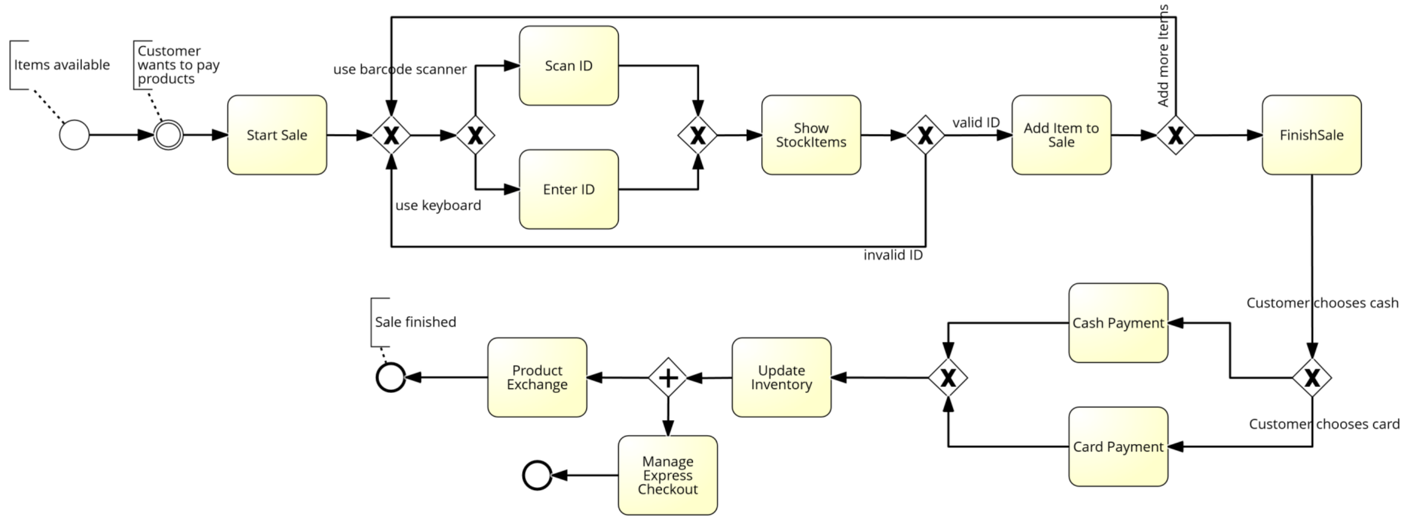

Figure A.6.: Control Flow UC1 - Start Sale

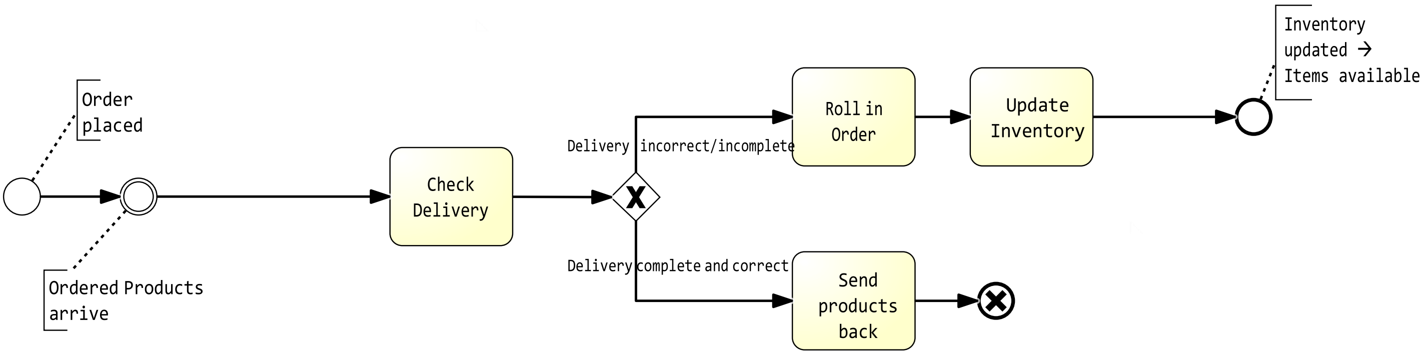

Figure A.7.: Control Flow UC4 - Receive Ordered Products

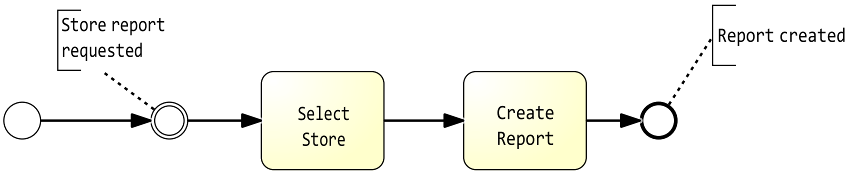

Figure A.8.: Control Flow UC5 - Show Stock Reports

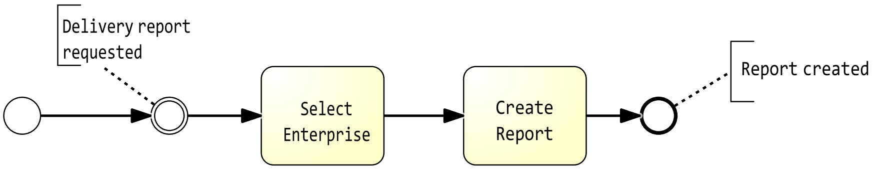

Figure A.9.: Control Flow UC6 - Show Delivery Reports

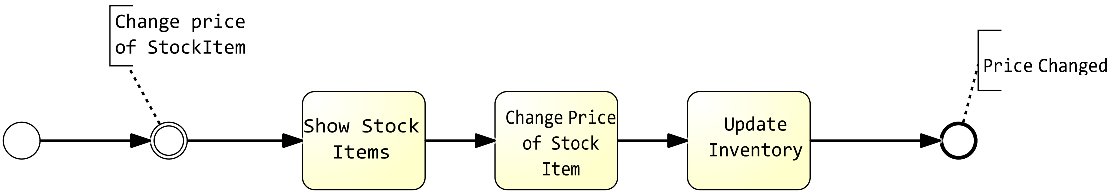

Figure A.10.: Control Flow UC7 - Change Price

A.3. Data Flow Diagrams

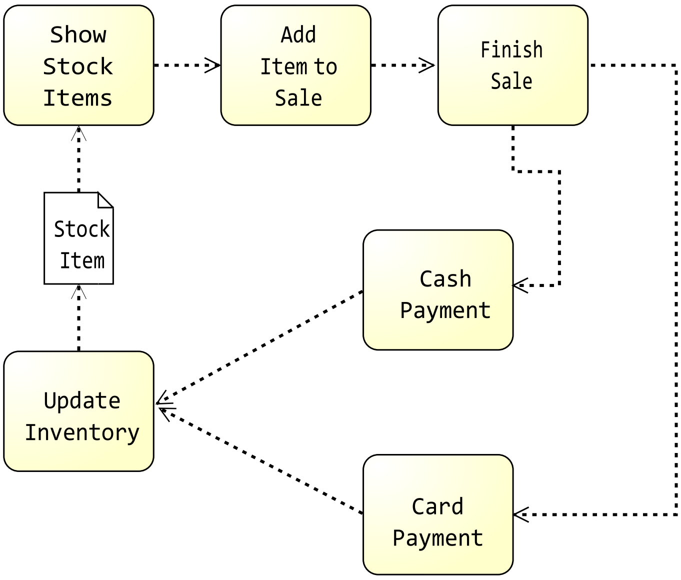

Figure A.11.: Data Flow UC1 - Start Sale

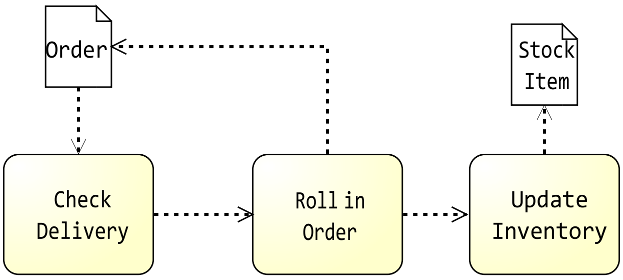

Figure A.12.: Data Flow UC4 - Receive Ordered Products

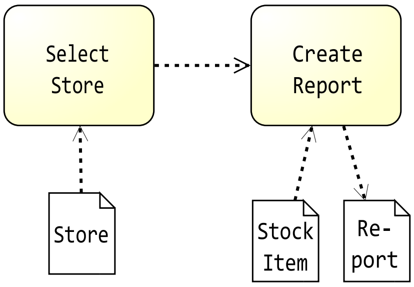

Figure A.13.: Data Flow UC5 - Show Stock Reports

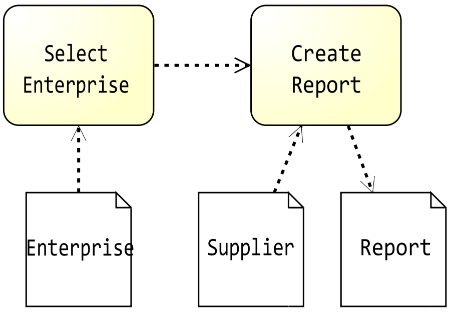

Figure A.14.: Data Flow UC6 - Show Delivery Reports

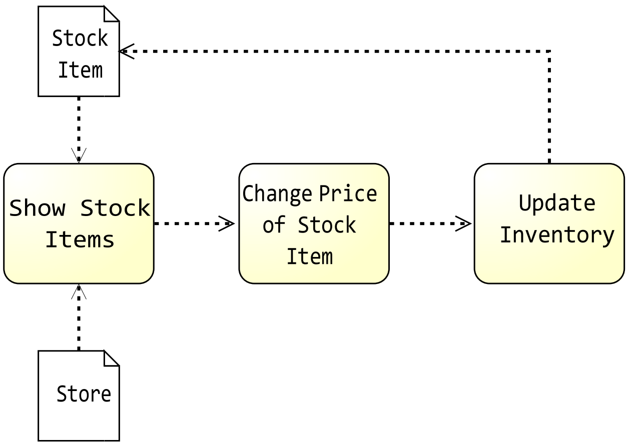

Figure A.15.: Data Flow UC7 - Change Price