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:	Prof. Dr.-Ing. Anne Koziolek
Advisor:	Dr. Robert Heinrich

01. January 2019 – 31. April 2019

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

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 (re-)designed their applications using the microservice architecture. The key challenge is to find an appropriate partition of the (legacy) application - 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 applications 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 graph-based identification approach using clustering techniques. Starting with business processes, the approach uses control flow and data flow dependencies to build two weighted graphs. From that, clustering techniques identify high cohesive sets of activity clusters on the one hand and data object clusters on the other. Finally, those sets are matched to generate compound clusters of activities and corresponding data objects. Each compound cluster corresponds to a microservice candidate.
An evaluation demonstrates that the (semi-)automated approach identifies adequate microservice candidates which are similar to a manual decomposition but identified with less expertise.

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 Sammlung unabhängiger, in sich zusammenhängender, aber lose gekoppelter Services, die die Defizite zentralisierter, monolithischer Software überwinden. Einige bekannte Unternehmen haben bereits ihre (bestands-)Software als microservice-basiertes System (um-)gestaltet. Eine Schlüsselaufgabe dabei ist es, die richtige Aufteilung der (Bestands-)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. Jedoch wachsen Applikation mit der Zeit und werden zunehmend komplexer, sodass die Aufteilung von Systemen zunehmend herausfordernder ist. Diese Thesis stellt daher einen graph-basierten Ansatz vor, der mittels Clustering-Techniken Microservice-Kandidaten extrahiert. Der Ansatz basiert auf der Prozesssicht und stellt Kontrollfluss- und Datenflussabhängigkeiten als zwei separate gewichtete Graphen dar. Diese werden benutzt, um mittels Clustering-Techniken stark zusammenhängende Aktivitätscluster einerseits und stark zusammenhängende Datencluster andererseits zu identifizieren. Anschließend werden diese zwei getrennten Cluster-Mengen abgeglichen, um zusammenhängende Cluster aus Aktivitäten und zugehörigen Datenobjekten zu erstellen. Jedes Cluster entspricht dann einem Microservice.
Die Evaluierung zeigt, dass der (semi-)automatische Ansatz plausible Microservice-Kandidaten identifiziert, die einer manuellen Aufteilung ähnlich sind. Im Vergleich zu dieser, können die Microservices jedoch mit sehr wenig Fachkenntnis identifiziert werden.

Contents

List of Figures

List of Tables

1. Introduction

The monolithic software architecture is the traditional pattern to design software, in which functionality is bundled in one single, large application [13]. Although monoliths have their strength, like fast development and simple deployment, they become an obstacle when they grow in size and become more complex [37]. Incomprehensible code structure makes it difficult to add functionality, fix bugs and enable new software engineering approaches like Continuous Delivery and Continuous Deployment [38]. Besides, the rise of cloud computing requires a new architecture that can fully exploit the rich set of features given by the cloud infrastructure [29].
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 [41], the ability to use different technology stacks and independent deployment [4] and the efficient resource utilization in cloud environments [29] explain the usage of microservice-based applications by big companies like Google, Netflix [6], Amazon, eBay [13] and Uber [41].
This chapter presents the motivation for the topic and the contributions of this thesis.

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 [19]. To tackle this issues, software engineers started to decompose their system into modules and provide the functionality over the network as Web Services [21]. 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[4] state that the boundaries between modules in SOA are too flexible and the application results in "a big ball of mud". 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 [41]. Chen *et al.* consider the microservice architecture as a particular approach for SOA [13]. 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 [21]. There is no consensus about the relationship between microservices and SOA but they both share common characteristics, like reusability and orchestration of modules or neutral development technology. 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 highly flexible, maintainable and scalable microservice architecture [13]. Consequently, moving existing applications to a microservice landscape is an upcoming philosophy in academia and industry [3]. Besides the migration of existing system towards a microservice architecture, many greenfield systems[1] are already designed using the microservice architecture.
Nevertheless, decomposing a (existing) system in loosely coupled, fine-grained and independent microservices is a time consuming task that requires tedious manual effort [21] and is technically cumbersome [16]. So far, it is done mainly intuitively and relies on the experience of software architects and system designers. Hence, an approach to identify microservices is required.

1.2. Research Questions and Contributions

The microservice architecture is a fast rising approach to structure a system as a collection of highly cohesive but loosely coupled and independent services. Large applications are decomposed into small, independent microservices where each service can be independently scaled and deployed.
However, one of the biggest problems in designing a microservice architecture is to decompose an application into a suite of small services while keeping them loosely coupled and highly 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 [20] and service granularity [22] 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.* criticise a lack of systematic approaches to reduce the complexity of the extraction process [22]. Extracting microservices from monoliths or design a larger greenfield application as microservice-based system therefore requires tedious manual effort and can be very costly [41] [32]. In the following, we present Research Questions (RQ) to tackle the issue of microservice identification:

> **RQ1: How to identify microservices based on the system specifications?**
>
> The research question can be further divided into more specific sub-questions, where the first question covers the literature research, the second one addresses the construction of a new approach and the last one deals with the evaluation.
>
> > **RQ1.1: Which is an appropriate strategy to decompose a system into microservices?**
> >
> > To identify possible strategies, a literature research is conducted. Suitable strategies and approaches are compared based on criteria identified in the literature research.

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

RQ1.2: How to identify possible microservices without detailed knowledge and manual effort?

To that end, the most promising strategy identified in RQ1.1 is used as basis. Thereupon, an approach is elaborated that aims to reduce the complexity and manual work that has to be done when identifying microservices.

RQ1.3: What is the accuracy of the approach?

Research question RQ1.3 is tackled by comparing the result of the approach with the results of other approaches.

1.3. Thesis Outline

The thesis 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. Further this chapter introduces a specific use case notation and the business process modelling language *BPMN*[2].
- Chapter 3 introduces the running example *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. Second, the most promising strategies and approaches are described and further compared using adequate criteria.
- Chapter 5 proposes a graph-based approach to identify microservices. The process is divided in several steps, where each step is presented in a separate section.
- Chapter 6 applies the approach to the running example. All intermediate results will be presented as well as a decomposition of *CoCoME* into microservices.
- Chapter 7 evaluates the approach. First, the evaluation method is presented. Second, two reference sets are introduced which are used to evaluate the result of the previous chapter. Finally, the threats to validity are presented.
- The thesis is concluded by chapter 8, where the main outcomes are summarized and discussed. Eventually, the limitations and future work are discussed.

[2]Business Process Model and Notation

2. Background

This chapter introduces and compares the monolithic software architecture and the microservice architecture. Further, benefits and challenges of the microservice architecture are outlined. Also, it introduces a well known and established use case notation technique and a standardized notation to capture business processes.

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 [37]. 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 [16]. 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 [41]. Fast prototyping is supported by current frameworks and development environments (IDEs), which are still oriented around developing single applications [37]. But once they grow in size, they become exceedingly difficult to understand and hard to maintain without reasonable effort [41] [20]. A complex and large code base prevents a fast addition of new features and makes the application risky and expensive to evolve [28]. 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 [41]. Moreover, it is difficult to adopt newer technologies without rewriting the whole application, as monoliths are build on a specific technology stack [37] [32].
Scaling is only possible by duplicating the entire application, namely *horizontal scaling*. Consequently, large portions of the infrastructure remain unused, if only parts of the application need to be upscaled or even used [25] [19].
Rui Chen describes the shortcomings of monolith as follows: "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" [13].

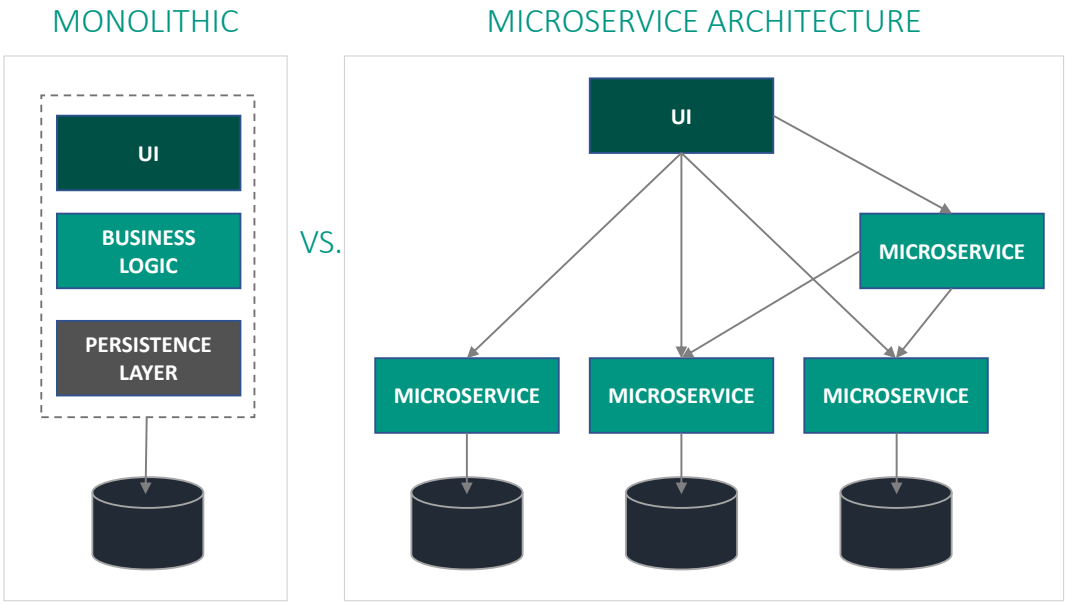

Figure 2.1.: Monolithic vs. Microservice Architecture

2.2. Microservices

M.Fowler and J.Lewis describe the microservice architectural style as "an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API"[18].

2.2.1. Definition

Yet, the term microservices is not formally defined in academia. A popular definition of microservices is "a collection of cohesive and loosely coupled components, where each service implements a business capability"[3]. Based on that definition, M. Amiri introduces three principles upon which the architecture is build: *Bounded Context, Size, Independence* [3].
The first principle is about related functionality, that is combined as a single business capability - the *bounded context* [41]. Each capability is implemented by one microservice where a business capability is something that a system does in order to generate value[35]. The *Size* of a microservice is defined by the number of features it provides (namely bundled functional capabilities) [26]. There is no consensus about the "proper" size of a microservice [35], but several guidelines exist: 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 [26]. However, the sizes vary from system to system [41] and even different sizes for each microservice in a specific system are possible [35]. The bottom line of *Independence* is expressed in Amiri's description of microservices as "a collection of highly cohesive and loosely coupled components" [3]. Highly 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 [13].
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 [41]. The design of microservices

mainly follows the *Single Responsibility Principle (SRP)*: Each service should not have more than one reason to change [17]. The SRP mainly corresponds to the idea of not implementing more than one business capability. The following paragraphs cover the benefits and challenges of the 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 [41]. 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 require a time consuming build of a new version and the redeployment of the entire application [18].

Availability, Resilience and Fault Isolation
Microservices are designed to operate independently of each other and to tolerate failure of services [18]. 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 [32].

Scalability and Resource Utilisation
Small and independent microservices allow more fine-granular horizontal scaling [26]. Single services can be duplicated to cope with changing workload during runtime [13]. Thus, dynamic (de-)allocation of resources on demand prevent infrastructure from being idle [16]. Scaling monoliths can only be attained by duplicating the entire application, leaving resources unused [20]. Further, each microservice is deployed on the best suitable infrastructure for its needs, allowing a more efficient system organization [37].

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 [33].
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 experts for UI-development, database architects, back-end engineers and project managers. This minimizes the need for communication and interaction between the teams and thus, speeds up the productivity. Ultimately, microservices enable a more agile flow of development and operation [20], 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 [41]. Microservices can be realized using different programming languages, technologies and even deployment environments [13]. 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 [21] [28].

2.2.3. Challenges

The previous section provides a vast amount of benefits that come with microservices. However, microservices are not the panacea of software engineering. System developers have to face challenges that can mitigate the benefits as described previously.

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 [19]. 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 [29]. Besides, the distributed microservice landscape complicates the logging mechanisms and performance monitoring [2]. Traditional centralized logging, as it is used in monolithic applications, is no 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 [18]. 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 [8].

Data Consistency
Distributed systems need to share data. Heinrich *et al.* propose two concepts for the database architecture [41]: 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. On the one hand, this approach overcomes the issue

of consistency, as data is stored centrally. On the other hand, sharing results in a loss of independence. Scaling can only be achieved through replicating the whole database. According to Tyszberowicz *et al.*, the first concept is preferred [41].

Decomposition
Decomposing a system into microservices is a very complex task that requires experienced system architects and domain experts [18]. It is irrelevant whether an existing system is decomposed into microservices or whether a new system is designed as a microservice-based application. The effort is very high in both cases.
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 [35]. On the other hand, too coarse grained microservices debilitate the scalability.

2.3. 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.* [14]).
An example is given in Table 2.1: 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 2.1.: Example Use Case in Tabular Form, Source: [24]

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 sorts of stakeholders (non-technical and technically experienced) are capable to provide the necessary information in terms of use cases.
However, the transformation into business models as presented in Sec.2.5 is not always trivial and requires some manual effort to produce high quality business processes. Nevertheless, it is necessary as the system requirements of the running example (cf. chapter 3) are given in form of textual use cases.

2.4. Business Process Model and Notation

The Business Process Model and Notation (BPMN) is a graph oriented language to describe business processes. Originally, BPMN was designed to describe activities and their control flow dependencies only [30]. Since the introduction of BPMN 2.0, it is possible to model the data needs and the data results of activities [11]. Consequently, BPMN is capable of expressing the control flow and to approximate the data flow of business processes [39]. In the remainder, BPMN and BPMN 2.0 is used interchangeably.
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.2.5 introduces a formal approach to generate BPMN processes from use case sets. Next, the BPMN 2.0 process definition is shortly introduced:

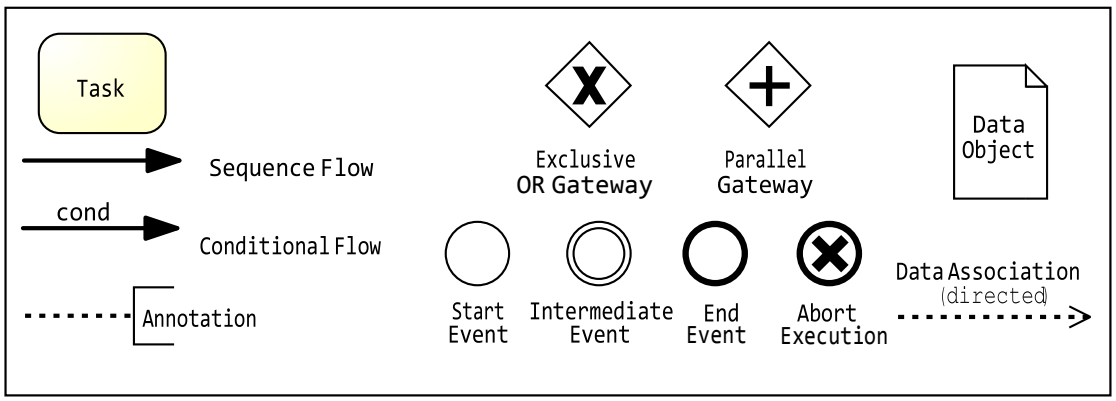

Figure 2.2.: BPMN Notation (Subset)

Fig.2.2 shows 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 gateway, 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

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

an *End Event*. In case of several branches (due to Gateways) stop events need to be placed at each end. *Intermediate Events* mark any other event that occurs during the process. The trigger for an event is modelled using the *Annotation* symbol. The *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, whereas producing data works in the opposite direction.

2.5. Transform Use Case Sets in 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" [30], 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 the work of Lübke [30], to transform use cases into BPMN models:

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 (cf. Fig.2.3)
3. Join the generated BPMN processes based on preconditions, trigger and postconditions (cf. Fig.2.4 and Fig.2.5)
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 phase only includes simple substitutions, as the *extend* and *include* relationships simply have to be replaced by the actual use case.
Given a use case with a precondition, triggers *1 to n*, steps *1 to n*, an extension that is based on a condition, extension steps, data objects that are accessed by the steps and a postcondition, phase number two represents the main part of the transformation.
Fig.2.3 is an additional illustrative diagram that explains this phase. First, the preconditions are 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 case's standard process is represented by a single task. This procedure includes some refactoring as further enlightened in the final phase. 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 model. Given these point, it is necessary to check if a data object is already referenced by a previous task. Jumps and alternatives

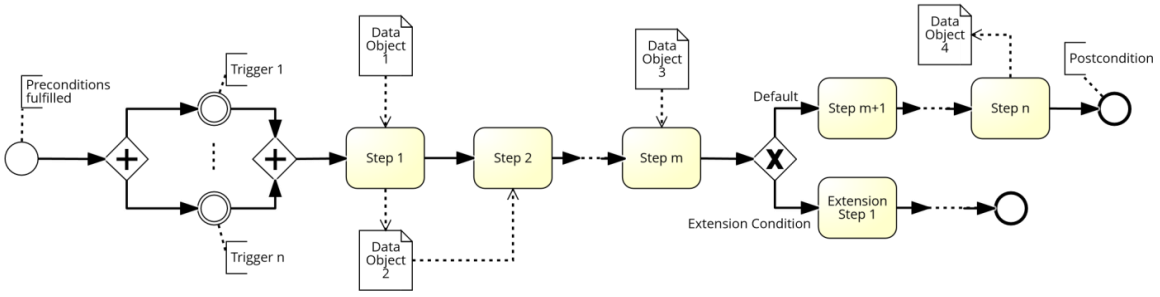

Figure 2.3.: Use Case transformed in a BPMN process

are modelled using Exclusive Or (XOR) Gateways. Finally, the postcondition is added as annotation and connected to the end event.
In the previous phase, each use case is transformed into a single, independent BPMN process. However, 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 A) 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.2.4).

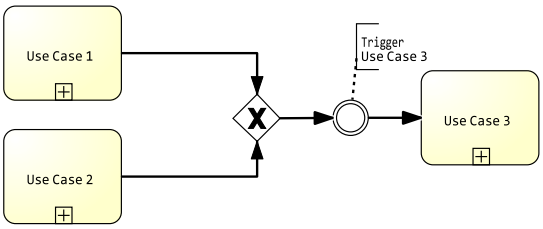

Figure 2.4.: Join multiple BPMN processes on same postconditions

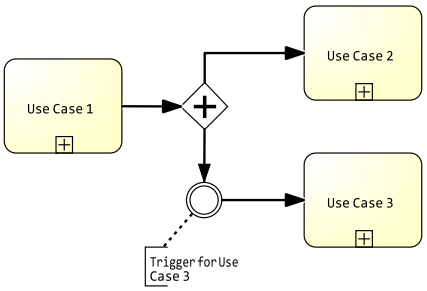

Figure 2.5.: Join multiple BPMN processes on same preconditions or triggers

If several use cases have the same preconditions or triggers and therefore use the same postcondition (Fig.2.5), join them by introducing a Parallel Gateway and split the control flow. Notice, that the use cases in Fig.2.4 and Fig.2.5 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 arcs to the remaining object.
Finally, all business models need to be refactored. Whereas the prior phases are very

algorithmic and follow a predefined concept, this phase 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, step one contains two activities and needs to be split in *Change Price* and *Update Inventory*.

2.5.1. Limitations and Drawbacks of the Transformation

In the previous section, a method to transform use cases in a set of BPMN models is introduced, as the running example only provides system specifications in form of detailed use cases (cf. chapter 3). 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. Hence, it is necessary to eliminate synonyms to ensure an unadulterated outcome.
In regard to tasks, it is indispensable to identify synonyms across all use cases and hence, the resulting business processes, in order to extract the structural dependencies properly. In case that synonyms are not detected, structural dependencies between processes like using the same functionalities would disappear and the business processes would end up being broadly independent from each other. In the course of the identification process (cf. chapter 5), this causes the tasks in the control flow graph to be arranged in a circle-like order, so that the resulting clusters would each correspond to the single business processes. 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.

3. Running Example

The *Common Component Modelling Example (CoCoME)* is a case study on software architecture modelling [24][23]. 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.
A more detailed description of the CoCoME system can be found in the literature [24][23].

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. Herold *et al.* elaborated a fully detailed description[24].

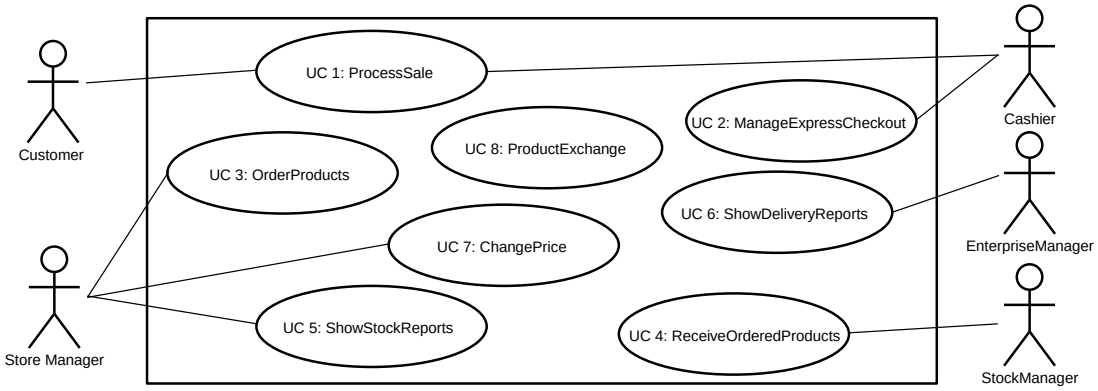

Figure 3.1.: Use Case Diagram of CoCoME

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 average 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 a 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" [41] was provided by the supervisor of this thesis. Besides, *"Service Cutter - A Systematic Approach for Service Decomposition"* [21] was cited by various papers, 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 eight most promising approaches in the area of microservice identification. Other papers like [42] only presented a conceptual train of thought, whereas [29], for instance, focuses on migrating strategies on infrastructural level. This thesis mainly focuses 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 eight criteria and explains why they take part in the comparison. The comparison itself is done by applying the criteria to each approach using the tables 4.2 and 4.3. Incidentally, the comparison including some criteria are inspired by the work of [20]. 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	Title	Author (Year)	Origin	Search String
[32]	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*
[41]	Identifying Microservices Using Functional Decomposition	S. Tyszberowicz *et al.* (2018)	*provided by supervisor*	-
[35]	Partitioning Microservices: A Domain Engineering Approach	I. J. Munezero *et al.* (2018)	ACM	*partition microservices*
[13]	From Monolith to Microservices: A Dataflow-Driven Approach	R.Chen *et al.* (2017)	IEEE	monolith microservice
[16]	Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems	A. De Alwis *et al.* (2018)	Google Scholar	identify microservices
[21]	Service Cutter: A Systematic Approach to Service Decomposition	M. Gysel *et al.* (2016)	[4]	-

Table 4.1.: List of authors and approaches

4.2. Approaches for Identifying Microservices

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

Extraction of Microservices from Monolithic Software Architectures
The approach presented in [32] 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 are weighted 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 Microservices
[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 reads and writes of each activity. Activities that are directly or indirectly connected and perform write or read operations are more likely to be partitioned 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.

Microservice 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.

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

Identifying Microservices Using Functional Decomposition
The approach presented in [41] identifies microservices by functional decomposition of the software requirements, provided as use case specifications. In order to achieve the decomposition, 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. Irrelevant nouns, verbs and synonyms are eliminated via brainstorming. 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. A relation exists, if a *system operation* reads or updates a *state variable*. Finally, the approach uses graph analysis tools to determine clusters, where each cluster is a potential candidate for a microservice that fulfils the criteria of low coupling and high cohesion.

Partitioning Microservices: A Domain Engineering Approach
Munezaro *et al.* [35] 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 determine the boundaries of each responsibility and define 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.* [13] use a top-down data flow driven decomposition approach to determine highly 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 consists 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 [16].
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 through e.g. analysing the log data), the information found is passed to the second part of the process. Algorithms process 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.* [21] introduce a service decomposition tool based on 16 coupling criteria derived 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 an user. To analyse and process the coupling criteria, *Service Cutter* creates a weighted graph. The nodes represent the nanoentities which are connected via weighted edges. The weighting corresponds to the sum of all scores defined by the criteria as mentioned above. In addition, the user can prioritize individual criteria by adding a multiplier that increases the weight of the score. In the end, an exchangeable clustering algorithm identifies potential microservice candidates where each cluster correspond to a highly 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: The **Basic Concept** recaptures the underlying approach of the microservice identification for classification purposes. The column **Prerequisites** present the necessary preconditions for the success of the approach. For example, the approach mentioned in [32] cannot be used without meaningful VCS[7] data. The **Input** row describes the type and amount of input that is used to realize the approach, i.e. Data Flow Diagrams in [13]. The row **Tool Support** indicates, whether the approach has been implemented or if other supporting tools are available to simplify the identification of highly 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. [32]), whereas others generate a predefined granularity (e.g. always the most fine-grained microservice candidates [16]). **Validation** compares how the approaches are validated to strengthen the credibility of the individual results. Further, each approach has 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 [32] is the result of a master thesis [33]. Therefore, the degree of available information is larger compared to other approaches. The algorithmic recommendation of microservice candidates is implemented in a web-based, open source prototype and permits to choose three different coupling criteria, which can be combined

[7] Version Control System

for better results. Nevertheless, the main limitation relies in its type of input data: meaningful VCS data. For instance, a developer must not change two independent, unrelated functionalities and commit the changes together as this would mistakenly indicate that these functionalities belong together. The developer rather needs to split independent alteration across different commits to not influence the outcome.

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 lacks formal explanation and needs 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. 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* [21]. 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.

Tyszberowicz *et al.* [41] use external tools to realize their approach. Once the operations and state variable are identified, the approach 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 ([41][18][19] and more), it requires the expertise and experience of domain experts.

Chen's semi-automate approach [13] is based on Data-flow Diagrams (DFD). Transforming the traditional DFDs to purified DFDs is not trivial and 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.* [16] 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

[8]https://www.cs.drexel.edu/ spiros/bunch/
[9]Project which lacks any constraints imposed by legacy systems
[10]Domain-driven Design

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 [21] 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 [32] and [13]. 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.

Approach/Criterion	Mazlami *et al.* [32]	Amiri [3]	Baresi *et al.* [4]	Tyszberowicz *et al.* [41]
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/gmazlfrontend)	Clustering tool Bunch	experimental prototype (https://github.com/mgarriga/decomposer)	uses external graph visualize and analysis 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 level, 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 [21]	case study, compared to three manual implementations
Limitation	needs 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.* [35]	Chen *et al.* [13]	Alwis *et al.* [16]	Gysel *et al.* [21]
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 [21]	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. Approach for identifying Microservices

This chapter presents an approach 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, shortcomings 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, which means that this type has to manage the identification process with less input compared to the other one.
The approach we propose is based on pre-existing system requirements. No existing implementation is used, consequently the presented approach is to be classified as greenfield approach.
In the following, the strategy on which the approach is based is presented. Subsequently, the approach is introduced and partitioned in several steps, where each step is presented in its own section.

5.1. Underlying strategy

To answer *RQ1.1*, eight suitable approaches to identify microservices were presented and compared in chapter 4, using well-defined criteria. Most of them are based on strategies that 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.
In contrast, the approach *Object-aware Identification of Microservices* proposed by M. Amiri [3] is based on a strategy 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 as introduced in Sec.5.2.2 and Sec.5.2.4. This leads to the fact that highly 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 this 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 solely based on data dependencies, ignoring any identified structural dependencies.
Although the approach reveals some weaknesses, the fundamental idea of Amiri, in our view, [3] is viable for the identification of microservices.

5.2. General Approach

For answering *RQ1.2*, this thesis proposes a graph-based microservice identification approach using clustering on control flow and data flow which is inspired by Amiri's work on *Object-aware Identification of Microservices* [3].

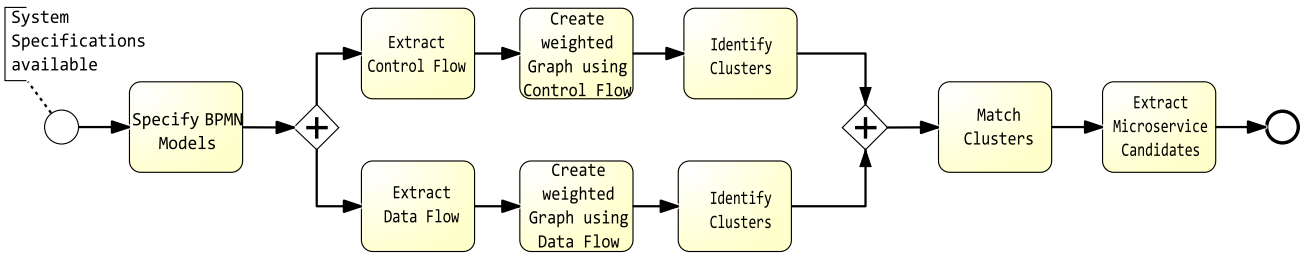

Figure 5.1.: Overview of the identification approach

Fig.5.1 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[3], 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 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.
RQ1.2 also adresses the subject of necessary know-how and the amount of manual effort. Regarding this, 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 not to be neglected, as the

extraction process, graph creation and cluster matching is not yet automated. Nevertheless, the structural process enables the implementation of the entire approach, excluding the BPMN model specification step. However, implementing an approach is beyond the scope of the thesis. In the following, each step as presented in Fig.5.1 is introduced in detail.

5.2.1. Specification of BPMN Models

Chapter 2 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 specifications are not directly given in form of business processes, but rather in the form of use cases, UML models, domain models or even as textual description in natural language. Therefore, the first step is to specify BPMN models using the input in form of 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" [11]. 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 2.5 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 [15]. Again, the tool discovers the models dynamically, using log files of a legacy system.

5.2.2. Extract Control Flow

Background: 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 highly 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.2.4). 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.

Process: All that has to be done to extract the control flow, 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. The control flow has to be extracted for each BPMN model that was specified in the previous step.

5.2.3. Create a weighted Graph using Control Flow

Background: In section 5.2.2, 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 highly cohesive functionality among all BPMN models, thus among the entire system. Fig.5.2 shows an the control flow of an exemplary BPMN model whose object-related information has already been deleted (cf. Sec.5.2.2).

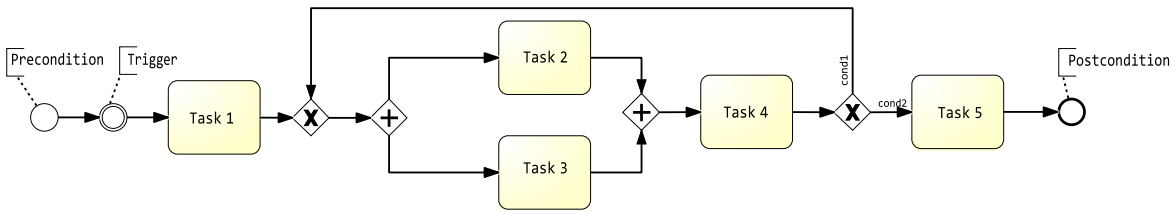

Figure 5.2.: An exemplary BPMN model illustrating only the control flow

Based on all BPMN models derived from the system specifications, a directed graph G is built. The vertices represent the tasks/activities in the BPMN models. The edges correspond to the identified dependencies based on the control flow. To visualize this step, for the sake of simplicity only one BPMN model is presented and transformed into a Graph. Fig.5.3 illustrates the Graph that corresponds to the BPMN model as depicted in Fig.5.2 and demonstrates the transformation. This graph would be bigger, if more BPMN models were used.

Process: 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 (parallel or XOR). This decision is based on the assumption, that two activities are more likely to be in the same microservice, if they are directly connected in a business process.
Regarding the weights, it is decided to assign a value of 1 to each edge, notwithstanding of the nature of the connection, which is either 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 occur in multiple models, this indicates a stronger cohesion. As a result, the edge in the graph that connects the tasks in question receives a greater weight (corresponding to the number of occurrences).

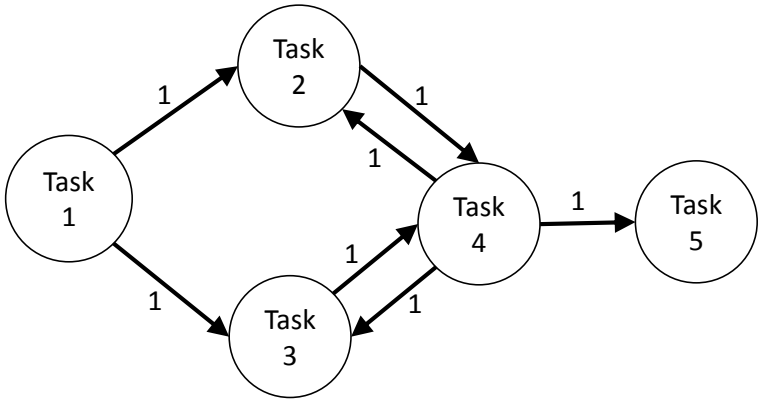

Figure 5.3.: Weighted graph using Control Flow Dependencies

5.2.4. Extract Data Flow

Background: 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. In order to achieve this, we propose to use clustering based on data flow to identify highly cohesive but loosely coupled set of data object clusters.
Usually, data flows are represented using specific notations of Data Flow Diagrams (DFD), i.e. a notation proposed by E.Yourdon [43]. For simplicity's sake, we relinquish to introduce another model notation and use the BPMN symbols instead.
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 2.0 extension provides the possibility to visualize an approximated data flow based on the data needs and writes of each activity. The data flow can only be approximated due to the capability of BPMN to express the data needs and the data results of single activities only, whereas the data flow describes the flow of data in a process. This problem can be explained using Fig.5.4 as an example:
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.

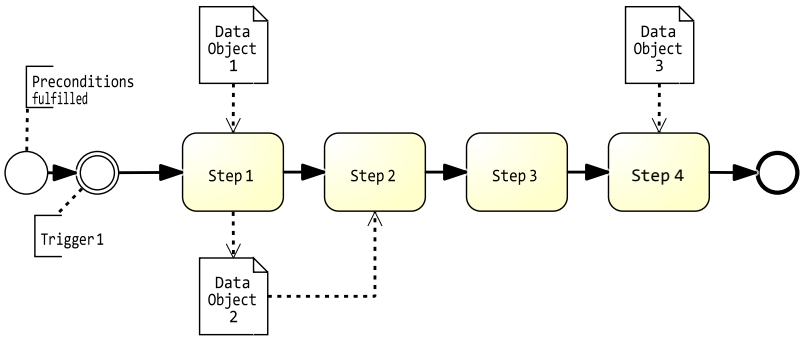

Figure 5.4.: BPMN model to illustrate the issue of the data flow approximation

However, the approach presented in this thesis aims to reduce the required expertise and the additional information that is necessary during the execution. Hence, it is fundamental to approximate the data flow based on the data needs and writes of each activity. As in this case, the data flow has to be approximated for each BPMN model.

Process: In the following, the process to extract and approximate the data flow from BPMN models is presented:
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 whom they might exchange data. In this case, 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. Fig.5.5)
- 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 and therefore not necessary for the data flow (cf. Fig.5.8)

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

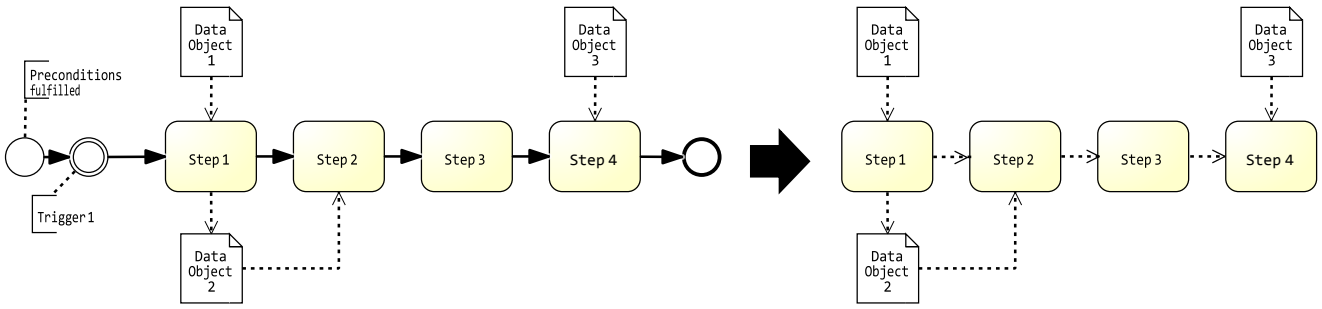

Figure 5.5.: Restore data flow connection

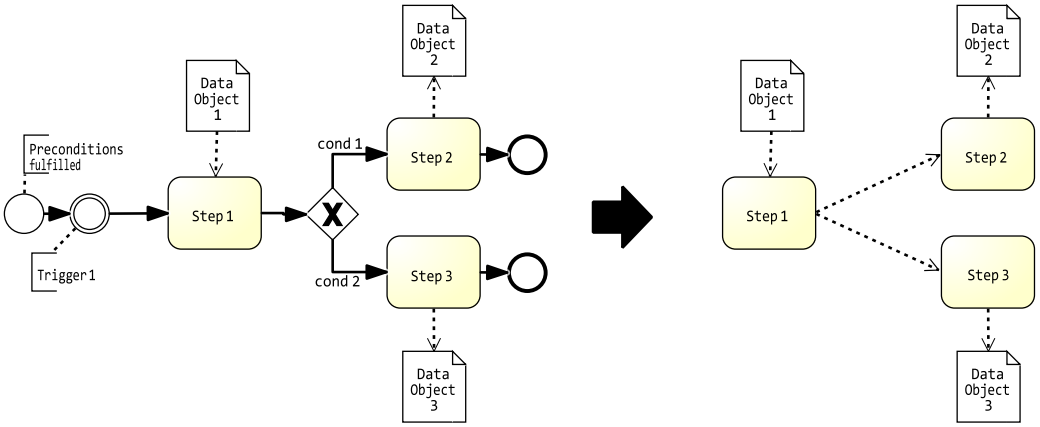

Figure 5.6.: Split data flow connection

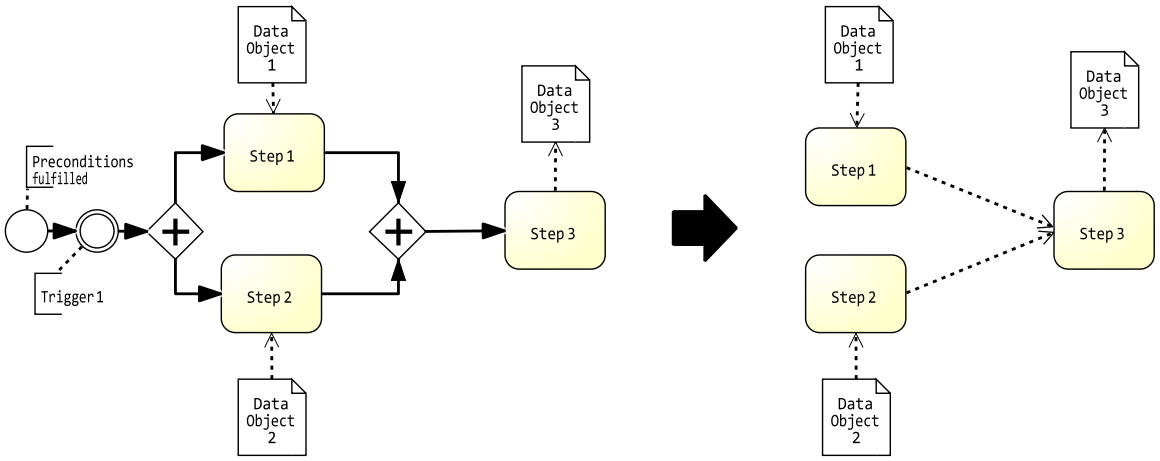

Figure 5.7.: Merge data flow connection

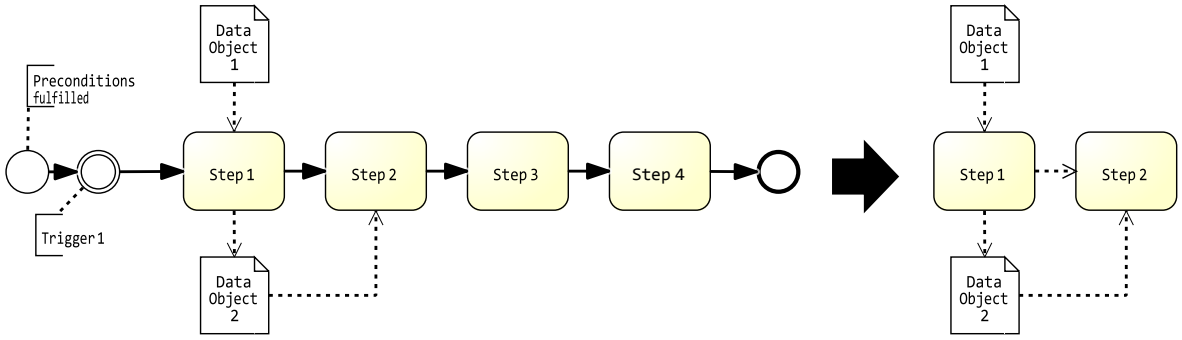

Figure 5.8.: Remove unnecessary tasks

5.2.5. Create a weighted Graph using Data Flow

Background: To identify highly 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 are read by the same task ii) a data object value is used while writing to (or creating) another data object.
Regarding i), 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, a pair of data objects represented by two vertices is to be connected by an edge in case both objects are read by the same task. 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.5.2.4). It is noticeable that it requires additional information to decide whether two data objects have one of the proposed connections. Fig.5.9 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.

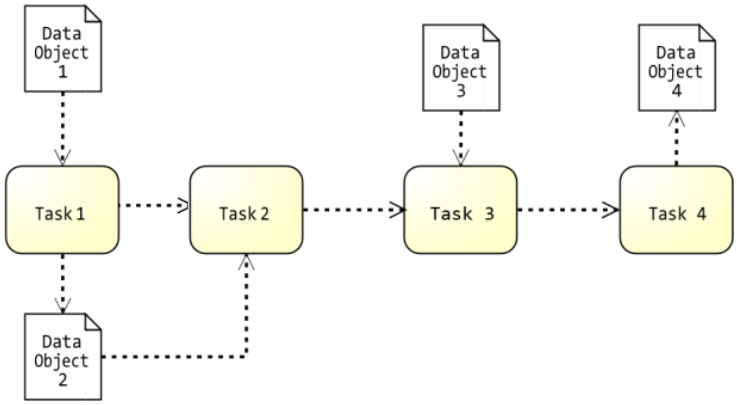

Figure 5.9.: Data Flow Diagram extracted from BPMN process

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 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 between them.
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*. Due to this, the identification of 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.

[1] $n \in [0..]$, where 0 represents neighbouring tasks
[2] Following the Data Flow Arcs when counting

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.
In the first option, data object dependencies are frequently underestimated, as no information flow from one task to another can be distinguished. Therefore, option one is also discarded.

Process: 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*, where n represents the maximum amount of tasks in between, has to be defined. Having *n=0*, only data objects that are processed by neighbouring tasks are considered to share a dependency and therefore are 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 are quite often distributed among several tasks. Fig.5.10 illustrates an example:

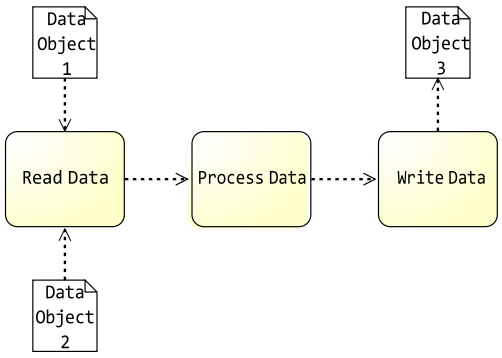

Figure 5.10.: Data Flow Diagram to demonstrate data processing and data storing

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 of *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 be not 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] and Tyszberowicz *et al.* [41], 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 dependency indicate a stronger cohesion. Fig.5.11 illustrates the Graph that is produced when applying the approach to the data flow described in Fig.5.9.

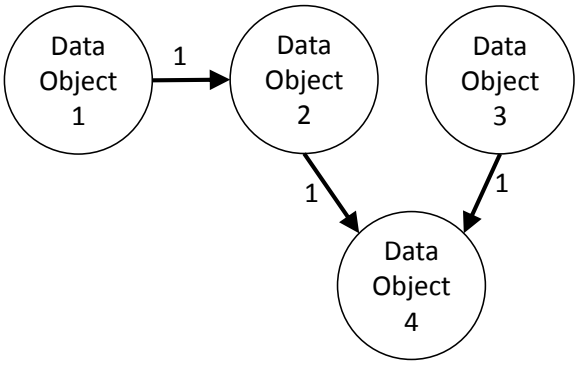

Figure 5.11.: Weighted graph using Control Flow Dependencies

5.2.6. Identifying Clusters

Background: 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 this thesis. Therefore, we use existing tools for the visualization and identification of clusters.

Process: 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

[3]https://gephi.org/

and repulsion called *Force Atlas* [7]. For the clustering, it uses a heuristic algorithm elaborated by Blondel *et al.* to find "high modularity partitions of large graphs" [10]. 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 it is in the case of CoCoME.
Upon further research, a tool called *Bunch* was chosen, which is a clustering tool that creates a graph decomposition by treating clustering as an optimization problem [31]. *Bunch* uses a genetic algorithm and a fitness function called *Turbo-MQ* [34]. 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" [34], in such a way, that it rewards intra-cluster coupling while penalizing inter-cluster coupling:

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

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 [27], which is a powerful graph description language. To visualize the clustered graph, we use the open-source tool *Graphviz* [4].

5.2.7. Cluster Matching

Process: 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. This decision is based on the same arguments that were used to weigh object dependencies(cf. Sec. 5.2.5). The calculation of data access between an activity cluster and a data object cluster is a trivial task and only requires examination of the BPMN models one more time: For each activity that accesses a data object, identify the

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

activity cluster the activity is located and the data object cluster the data is located. This is summarized 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. 5.12 provides an example.

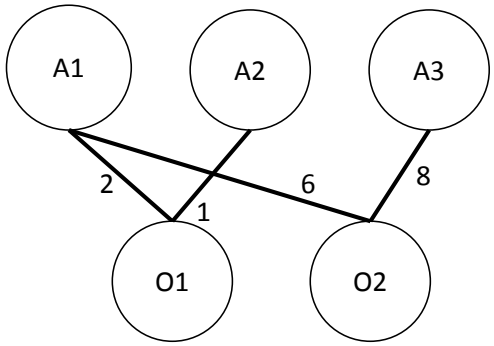

Figure 5.12.: Match Activity Cluster (A1-A3) and Object Cluster (O1-O2)

Speaking of the first approach, each data object cluster is simply matched to the activity cluster that is connected by the edge with the highest weight. Furthermore, object clusters that match the same activity cluster are merged. 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 straightforward and easy to apply, it has drawbacks: The cluster matching from data object point of view does not consider the overall dependencies. Regarding Fig.5.12, 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 the 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 regards to data access is the connected activity cluster node with the highest edge weight. In this case, the result obtained from Fig.5.12 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.5.2.6. Thus, highly cohesive activity and data object clusters are combined. In regard to the running example *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 approach to match clusters is not yet mature and only presented conceptual. Neither a concise definition is provided nor has it been tested on several case studies. However, we believe in the potential of the cluster matching method and propose a more detailed case study in this area. 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.

5.2.8. Extract Microservice Candidates

So far, BPMN models are used to extract the control flow and the data flow from a system's business processes. Based on heuristics, we determine dependencies between the activities and between the data objects and visualize them as two graphs to identify clusters of dense relationships that are weakly connected to other clusters. In the previous section, we propose different approaches to match the activity clusters and the object clusters in order to obtain combined clusters. 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. The administration also

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 CoCoME which is introduced in Chapter 3.

6. Application of the Approach

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 CoCoME, whose system specifications are defined in chapter 3. Speaking of the order, this chapter follows the process overview illustrated by Fig.5.1.

6.1. 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* [23]. 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.6.1 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 2.5. Apart from Fig.6.1, 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.

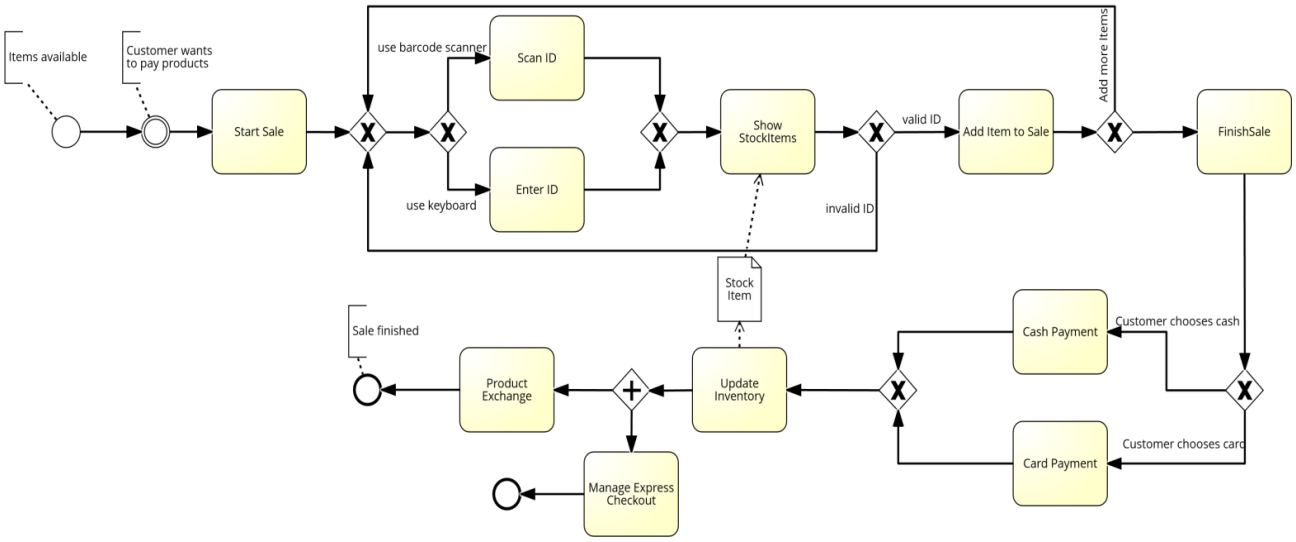

Figure 6.1.: UC1 - Process Sale (with UC2 and UC8)

6.2. Extracted Control Flow

In this step, the control flow is extracted from the BPMN models. Sec. 5.2.2 provides the detailed extraction process, but in a word, everything but the control flow elements are deleted. Fig.6.2 presents the control flow for *UC 3*. The remaining control flow diagrams are available in the appendix (cf. A.2).

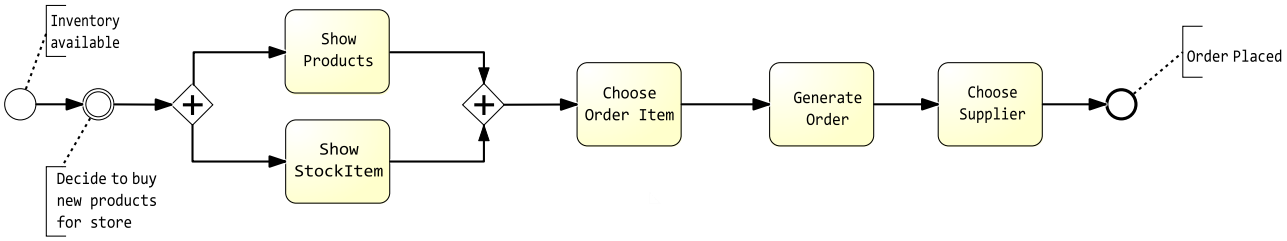

Figure 6.2.: Control Flow UC3 - Order Products

6.3. Extracted Data Flow

Like the previous step, the data flow is extracted from the BPMN models as described in Sec.5.2.4. Fig.6.2 presents the data flow for *UC 3*. The remaining data flow diagrams are available in the appendix (cf. A.3).

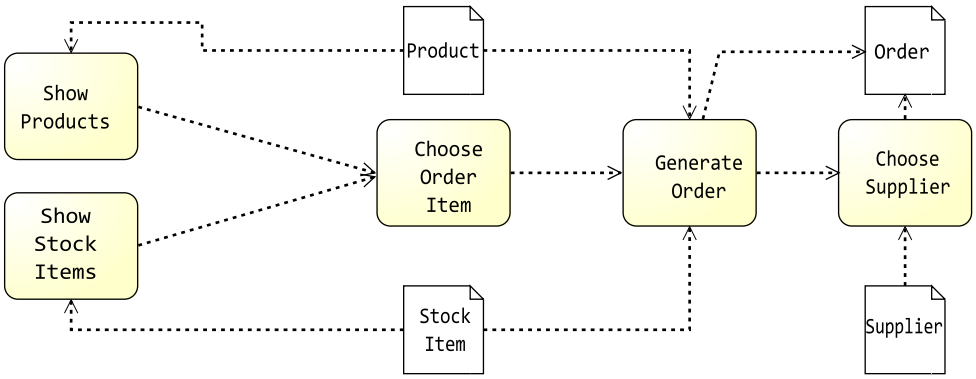

Figure 6.3.: Data Flow UC3 - Order Products

6.4. Control Flow Graph

Fig.6.4 illustrates the control flow graph of CoCoME which is derived from the control flow diagrams.

Figure 6.4.: Control Flow Information as Graph CoCoME

6.5. Data Flow Graph

Fig.6.5 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.

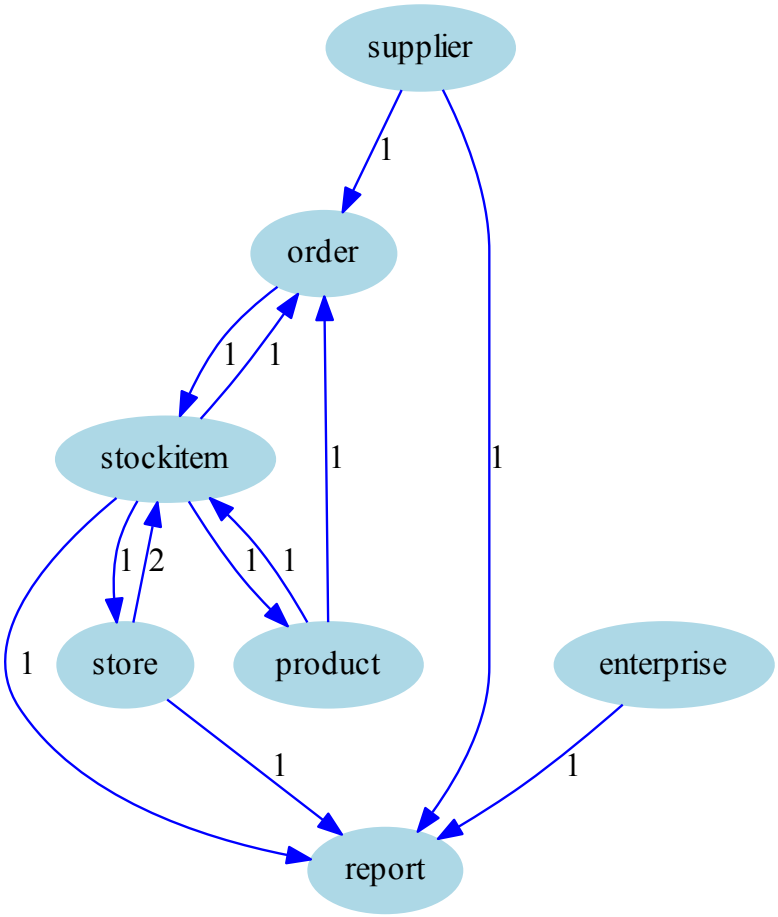

Figure 6.5.: Data Flow Information as Graph CoCoME

6.6. Activity Clusters

Fig.6.6 shows the activity clusters that are identified using the clustering algorithm described in Sec.5.2.6. The naming is inspired by the implementation of CoCoME [9].

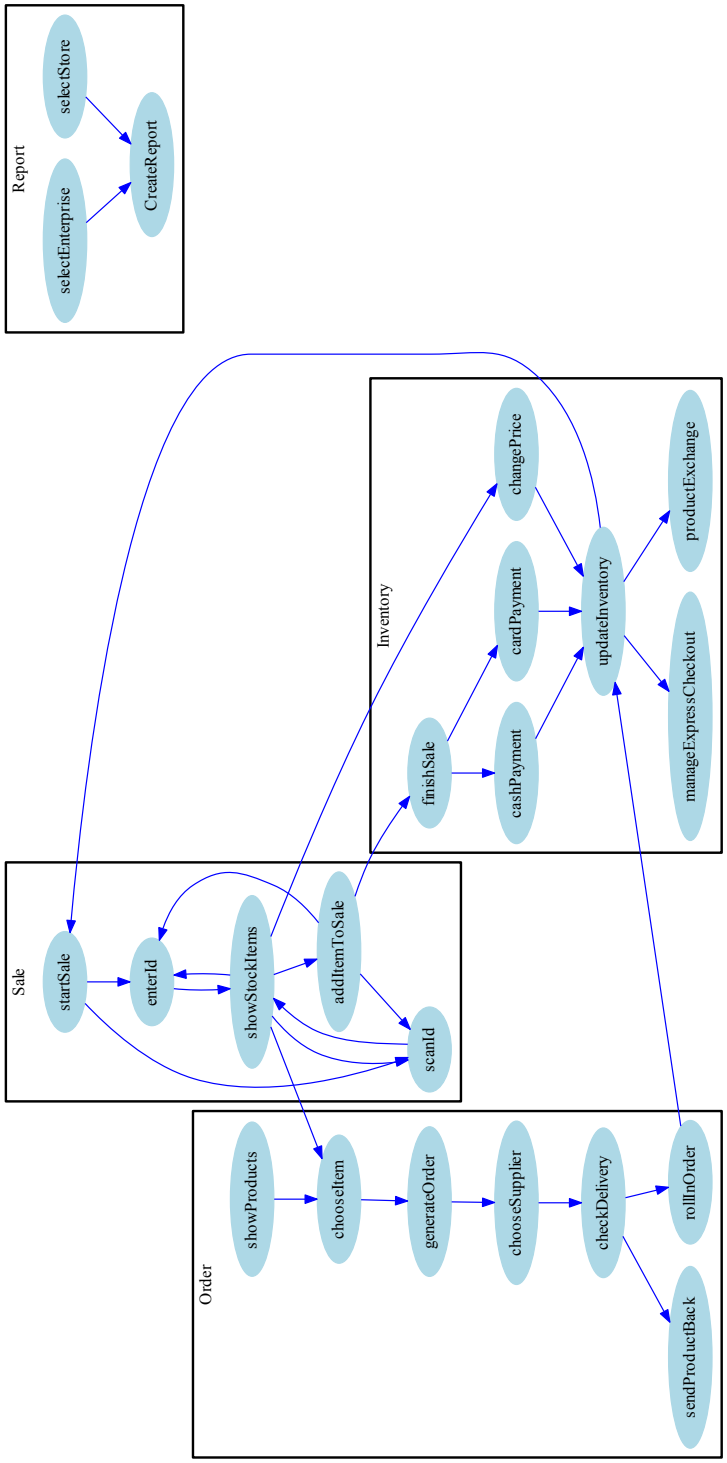

Figure 6.6.: Clustering on CoCoME's Activities

6.7. Data Object Clusters

Fig.6.6 shows the data object clusters that are identified using the clustering algorithm described in Sec.5.2.6.

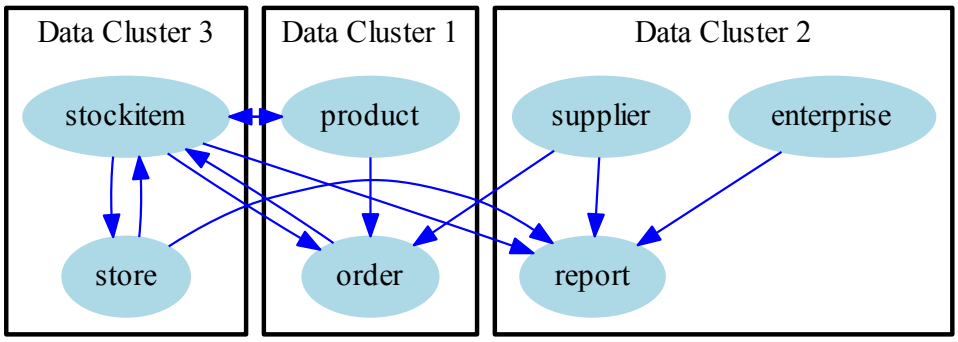

Figure 6.7.: Clustering of CoCoME's Data Objects

6.8. Matching of Clusters

In the following, the data object clusters and the activity clusters are matched. Section 5.2.7 provides several solutions, including one that is only given conceptional and still raises too many uncertainties. Consequently, the matching is done by applying the black box clustering approach. Hence, the data access dependencies between data object and activity cluster need to be elaborated first.

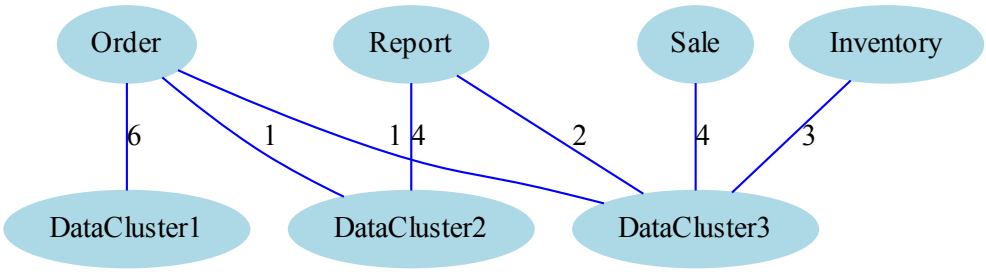

Figure 6.8.: Data Access Dependencies between Data Object Clusters and Activity Cluster of CoCoME

Fig.6.8 illustrates the data access dependencies between the four activity clusters *Order, Report, Sale* and *Inventory* and the *DataClusters 1-3.* The weights correspond to the amount of read and write accesses between activities and data objects within the corresponding clusters.
Finally, the clustering method introduced in Sec.5.2.6 identifies cohesive clusters of activity cluster nodes and data object cluster nodes.

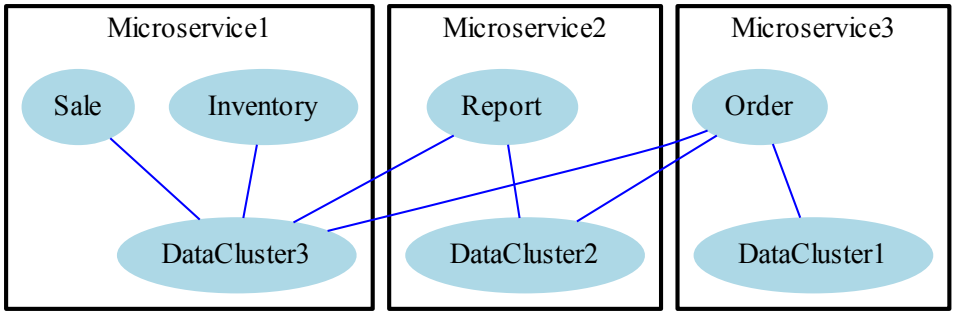

Figure 6.9.: Proposed Microservice Decomposition of CoCoME

6.9. Extract Microservice Candidates

In the previous step, cohesive clusters of activity cluster nodes and data object cluster nodes were identified. Each of the combined clusters correspond to a microservice candidate. Fig.6.9 illustrates the microservices that are identified. To compare the outcome with other results, it is necessary to elaborate the functionality that is provided by each microservice as well as the accompanying data objects. To be able to compare it with the reference sets, the functionalities must be renamed and abstracted. For example, *startSale* and *finishSale* become *Handle sale* or *enterId* and *showStockItems* become *Identify stock items.*

Microservice 1

- Handle sale
- Handle payment
- Manage express checkout
- Exchange products
- Identify stock items
- Handle inventory
- Change price
- **Data objects**: stockItem, store

Microservice 2

- Create delivery report
- Create stock report
- **Data Objects**: supplier, report, enterprise

Microservice 3

- Show products
- Create orders
- Handle deliveries
- Show suppliers
- **Data Objects**: product, order

7. Evaluation

To answer *RQ1.3*, a *Goal Quality Metrics Plan* (GQM) is to be created that specifies the key aspects of the evaluation. Chapter 6 applies the identification approach to the running example. As a result, a set of microservices is identified where each microservice provides some functionalities and administers data objects.
In order to evaluate this approach, we compare those results to two reference sets. In doing so, not only is it checked whether the right services have been found, but also whether the functionality and the data objects have been divided up correctly. During the evaluation, those aspects are examined independently. Functionality in this sense is a service offered by a microservice that is comprehensible and understood by non-technical stakeholders.
The rest of the chapter is structured as follows: First, the evaluation goals and metrics are introduced. Second, the design of the evaluation is demonstrated, including the presentation of two reference sets which are used to compare the output of the approach, when applied to the running example. Third, the evaluation results are presented. Second to last, the evaluation results are discussed. Eventually, the threats to the validity are outlined.

7.1. Evaluation Goals and Metrics

7.1.1. Evaluation Goals

Basili *et al.* originally proposed the *GQM Plan* (Goal Quality and Metrics) as a paradigm in software engineering to create specific quality models [5].
The main purpose of the *GQM Plan* is to identify the right metrics to assess the quality of an object in a particular environment. This should prevent the gathering of unnecessary metrics and measurements and consequently reduce the expenditure of work.
The *GQM Plan*, as an appropriate method to structure an evaluation, is used to define suitable goals, questions and metrics, which is why it is the means of choice in this evaluation.
The *GQM Plan* is a Top-Down approach and is 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. Second, 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 the consecutive evaluation is detailed out:

- **G1:** Determination of the accuracy of the approach to demonstrate that it is capable to identify adequate microservice candidates.

- **G1.Q1:** What is the *Precision and Recall* of the identified microservices compared to the reference sets?
- **G1.Q2:** What is the *Precision and Recall* regarding the functionality of the identified microservices compared to the reference sets?
- **G1.Q3:** What is the *Precision and Recall* regarding data objects each microservice administers compared to the reference sets?

7.1.2. Evaluation Metrics

Using metrics is mandatory to measure the quality of the elaborated approach. In this case, it is required to choose a metric that is capable of classifying a set of instances regarding their relevance. In regard to the following evaluation, those instances are either microservices, functionalities offered by microservices or data objects administered by microservices. Two reference sets are available as further depicted in Sec.7.2.2.
A metric that is capable to measure the relevance of a set of instances compared to a reference set is *Precision and Recall*. Subsequently, the proposed metric is briefly presented. *Precision and Recall* is a classification metric that measures the relevance of retrievable items with respect to a reference set [12]. 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 exactly 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 7.1.: Retrieval Matrix, Source: [12]

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.

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

In both cases, the values range from zero to one, where a higher value represents a more satisfying value in terms of completeness and purity of the retrieval. 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 7.1, the reference set used forms the relevant items, or N_{rel}. Accordingly, N_{ret} constitutes the retrieved items which are either the actual microservices, the functionality provided by the microservices or the data objects administered by the microservices. The remaining parts which are the non-relevant and non-retrieved items ($N_{\overline{ret} \cap \overline{rel}}$) are unimportant. The following list draws the analogy between Table 7.1 and the predominant evaluation scenario:

- **True Positives:** $N_{ret \cap rel}$
  - Identified microservices that have a similar partner in the reference set **OR**
  - Identified functionality that is assigned to the appropriate microservice **OR**
  - Identified data objects that are assigned to the appropriate microservice
- **False Positives:** $N_{ret \cap \overline{rel}}$
  - Identified microservices that do not have a similar partner in the reference set **OR**
  - Identified functionality, that is assigned to the wrong microservice **OR**
  - Identified data objects, that are assigned to the wrong microservice
- **False Negatives**: $N_{\overline{ret} \cap rel}$
  - Microservices in the reference set that are not discovered by the proposed approach **OR**
  - Functionality in the reference microservices which is not discovered and allocated by the proposed approach **OR**
  - Data objects in the reference microservices which are not discovered and allocated by the proposed approach
- **True Negatives**[1]: $N_{\overline{ret} \cap \overline{rel}}$
  - Microservices that are neither discovered by the approach, nor part of the reference set **OR**
  - Functionality that is neither distributed into a microservice, nor part of any microservice of the reference set **OR**
  - Data objects that are neither distributed into a microservice, nor part of any microservice of the reference set

Since the *Precision and Recall* metric not only gives information about how many microservices, functionalities and data objects have been identified in total, but also about how

[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.

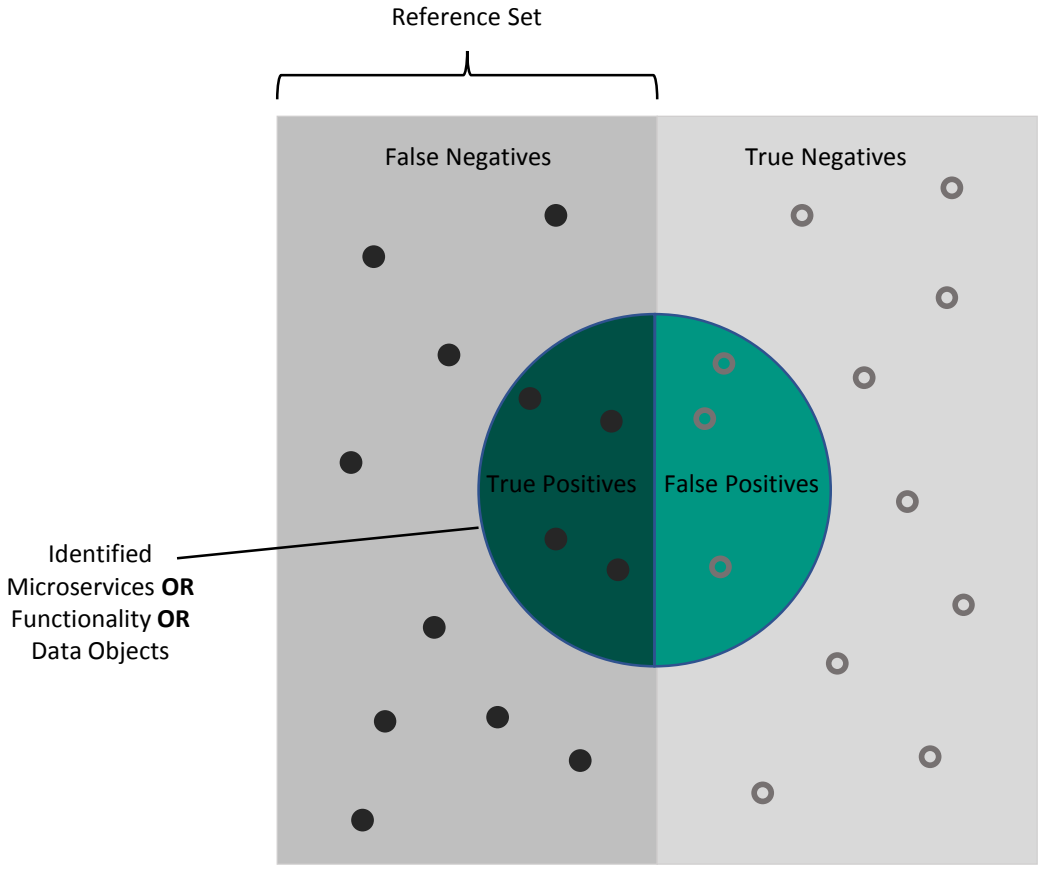

Figure 7.1.: Precision and Recall for Microservices

many of them are correct, it is well suited to determine the accuracy of the result compared to a reference set.
Hence, it is necessary to determine the *Precision and Recall* of the identified microservices, the identified functionalities and the identified data objects to answer **G1.Q1-G1.Q3**.

7.2. Evaluation Design

Evaluations can be structured and conducted in several ways. It is therefore necessary to specify the type of evaluation. This is followed by the presentation of the two reference sets.

7.2.1. Evaluation Setup

The primary goal of the evaluation is to determine the accuracy of the elaborated approach in order to answer **RQ1.3**. To achieve this, an *Outcome Evaluation* is an appropriate method

[40], as it shows how effective the approach is in terms of the expected results which are in this case adequate microservices. Therefore, the approach is applied to *CoCoME* and the outcome is compared to two reference sets. This is a well established procedure to reason about the accuracy of the approach.
CoCoME was initiated in a GI Dagstuhl research seminar as a community case study for software architecture, modelling and analysis. In recent years, it has been applied in various areas as a demonstrator for software evolution methods[23]. *CoCoME* is structured like a typical, distributed and component-based information system and is therefore well suited as a demonstrator for this new type of software evolution - the migration to a microservice-based software architecture.

7.2.2. Reference Sets

To evaluate the approach, the identified set of microservices (cf. Sec.7.4) is compared to two alternative decompositions of CoCoME: First, a decomposition proposed by an approach which identifies microservices by using functional decomposition[41] and second, a set of microservices which we manually identified[9].
In both cases the microservices are listed together with the functionalities they offer and the data objects they manage. Functionalities and data objects are listed behind the bullet points.

Reference Set 1: Functional Decomposition Approach
In *Identifying Microservices Using Functional Decomposition* [41], a systematic approach to find an appropriate division of a system into microservices is presented. It also uses CoCoME to evaluate the approach it provides.
As mentioned in Sec.4.3, the compulsory and non-trivial revision of nouns and verbs to eliminate synonyms etc. is a substantial disadvantage.
Regarding the evaluation, Tyszberowicz *et al.* claim that their approach identifies good microservices 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, a similar set of microservices. The third group identified a more detailed decomposition of CoCoME, but a revision reveals that the additional microservices are only a refinement of the proposed microservices.
However, the evaluation lacks profundity: Microservices are only compared on a more abstract level, since it is not checked whether functionality and data objects are assigned to the correct microservice. The following microservices are identified:

Sale

- Handle Sale
- Handle Payment
- Identify stock Item
- **Data objects**: -

ProductList

- Create orders
- Change price
- **Data Objects**: product

StockOrder

- Create stock report
- Handle inventory
- **Data objects**: stockItem, order

Reporting

- Create delivery report
- Handle deliveries
- **Data Objects**: -

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 finished before the literature review for the thesis started. Moreover, we were not aware of the microservice decomposition proposed by Tyszberowicz *et al.* [41] by the time we identified possible microservice candidates. Therefore, the process was non-biased.
The identification process itself was conducted manually and supported by the previous knowledge of the CoCoME domain. Once more, the time consuming and difficult identification process clarified the necessity of a structured and formal approach to identify microservices. Beside the use case specification, we used a monolithic implementation of CoCoME (the Hybrid Cloud-based Variant [23]) as additional information resource to discover requirements, functionality and dependencies. Subsequently, the system was decomposed into loosely coupled and highly cohesive microservices.
The following four microservices were identified:

Stores- and Sale

- Handle sale
- Handle payment
- Manage express checkout
- Exchange products
- Identify stock items
- Handle inventory
- Change price
- Handle stores
- Handle enterprises
- Handle deliveries
- **Data objects**: stockItem, store, enterprise

Product

- Show products
- Create products
- Show suppliers
- Create suppliers
- **Data Objects**: product, supplier

Order

- Show orders
- Create orders
- **Data objects**: order

Reports

- Create delivery report
- Create stock report
- **Data Objects**: report

Differences between both sets
The reference sets contain the same amount of microservices. Apart from the naming, those services are similar: One is responsible for the sale process, another one handles products, the third one is responsible for the orders and the last one is in charge of the reporting.

Nevertheless, the two differ from each other in regard to the functionality and data objects. Obviously, *Reference Set 1* features nine functionalities and three data objects whereas *Reference Set 2* contains 18 functionalities and seven data objects. It is important to note that all functionalities and data objects of the first reference set are part of the second one. Since the second reference set corresponds to a real implementation, it can be assumed that the first set is not complete.
Furthermore, both sets differ in the distribution of functionality and data objects among services. For instance, in *Reference Set 1* the data object *stockItem* is divided into the order microservice, whereas in *Reference Set 2* it is divided into the sale's microservice. To provide another example, the functionalities *Create Stock Report* and *Create Delivery Report* are distributed differently: In the first reference set, *Create Stock Report* is part of the *Report* microservice and *Create Delivery Report* is part of the order microservice whereas in the second reference set, both functionalities belong to the report service, which is more reasonable from the domain point of view. This example indicates a discrepancy between *Reference Set 1* and *Reference Set 2* where the latter corresponds to a real implementation. Thus, a stronger focus is placed on the comparison with the second reference set.

7.3. Evaluation Results

In this section, the precision and recall metric (cf. Sec.7.1.2) for the results of the approach (cf. Sec.6.9) is calculated using the reference sets described in Sec.7.2.2. For each reference set, the precision and recall instances are i) identified microservices, ii) identified functionality offered by a microservice and iii) identified data objects administered by a microservice.

7.3.1. Reference Set 1

Microservices The approach identified three services, *Microservice 1* is similar to *Sale* and *Microservice 2* is comparable to *Reporting*. *Microservice 3*, on the other hand, can neither be clearly assigned to *StockOrder* nor to *ProductList*, since it unites the functionalities of both. More specifically, *Microservice 3* is a more coarse-grained microservice that consists of *StockOrder* and *ProductList*. Therefore it can be concluded that three out of four microservices were successfully identified, and none of the identified microservices are incorrect.

$$Recall_{microservice} = \frac{3}{4} = 0.75 \qquad Precision_{microservice} = \frac{3}{3} = 1$$

Functionality Since *Microservice 3* consists of the microservices *StockOrder* and *ProductList*, functions that appear in this microservice and in either of the other two are regarded to be successfully identified. With this in mind, the results are as follows: The reference set counts nine functionalities. The approach identified 13 functionalities, from which five are assigned to the right service. Hence, the precision and recall values are:

$$Recall_{functionality} = \frac{5}{9} \approx 0.56 \qquad Precision_{functionality} = \frac{5}{13} \approx 0.38$$

Data Objects Regarding *Microservice 3*, the same line of reasoning is applied to the identified data objects: The reference set counts three data objects. The approach identified seven, from which 2 are assigned to the right service. In this case, the precision and recall values are:

$$Recall_{dataObject} = \frac{2}{3} \approx 0.67 \qquad Precision_{dataObject} = \frac{2}{7} \approx 0.29$$

7.3.2. Reference Set 2

Microservices As described in Sec.7.2.2, both reference sets contain similar microservices, which differ only by the names at microservice level. The functionalities and data objects, however, are different. Hence, the precision and recall value are identical in regard to identified microservices:

$$Recall_{microservice} = \frac{3}{4} = 0.75 \qquad Precision_{microservice} = \frac{3}{3} = 1$$

Functionality Again *Microservice 3* is considered to be a coarse grained union of the microservice *Product* and *Order*. In this case, the reference set counts 18 functionalities. The approach identified 13 functionalities, from which 12 are assigned to the right service. Hence, the precision and recall values are:

$$Recall_{functionality} = \frac{12}{18} \approx 0.67 \qquad Precision_{functionality} = \frac{12}{13} \approx 0.92$$

Data Objects The reference set counts seven data objects. The approach identified seven, from which 5 are assigned to the right service. In this case, the precision and recall values are:

$$Recall_{dataObject} = \frac{5}{7} \approx 0.71 \qquad Precision_{dataObject} = \frac{5}{7} \approx 0.71$$

7.4. Discussion of the Evaluation Results

The measurement of the precision and recall metric in the previous section gives information about the accuracy of the approach. Each of the identified microservices is also represented in the reference sets, which leads to a maximum precision of *1.0*. Due to the coarse-grained *Microservice 3* the recall is at 0.75. Additionally, it has to be noticed that the approach does not produce any incorrect microservices. Regarding the functionality and data objects, the comparisons to the reference sets produces different values for recall

and precision:

Compared to the first reference set, the approach produces less satisfying values. Only about half (Recall = 0.56) of the functionality in the reference set is correctly identified. In addition, only five out of 13 identified functionalities are assigned to the right service which leads to a small precision of 0.38. Regarding the data objects, the approach identifies two out of three correctly with an overall identification of seven data objects. Hence, the recall is at a satisfiable level of 0.67 but the precision is very low at 0.29. However, these non-satisfying values can be explained by the constitution of *Reference Set 1* as described in Sec.7.2.2. The discrepancy between this reference set and a comparable implementation (*Reference Set 2*) can explain the low recall value of the functionality. The incompleteness in regard to the data objects and the functionality explains the low precision values, as the approach identifies functionality and data objects which are not part of the reference set but obviously part of the CoCoME domain.

Compared to the second reference set, the values for precision and recall are much better. Almost every identified functionality is assigned to the right microservice, which leads to a precision value of 0.92. Obviously the *Reference Set 2* contains additional functionality that is not identified by the approach. As a consequence the recall value is not as high as the precision. However, it should be noted that the second reference set was created by using additional sources of information such as the source code whereas the approach only uses the provided use cases. With respect to the data objects, both precision and recall are at a satisfying value of 0.71. Finally, it has to be noticed that the approach does not identify any functionalities or data objects that are not part of *Reference Set 2*. Only a few instances are assigned to wrong services.
Since a stronger focus is placed on the comparison with the second reference set, four our needs the accuracy of the elaborated approach is satisfying.

7.5. Threats to Validity

To assess the overall validity of the evaluation results it is necessary to asses threats to the internal and external validity.

7.5.1. Internal Validity

Internal validity describes to what extend the evaluation allows unfounded results, usually as a result of systematic errors and bias [36].
First of all, *Reference Set 2* is based on our own implementation. We do not claim that this implementation represents the best or the only way to decompose CoCoME into appropriate microservices. However, the identification process was based on a sufficient knowledge about the CoCoME domain and the microservice topic. Hence, we claim that *Reference Set 2* can be used to determine the accuracy. However, to eliminate final doubts, either another independent reference set should be consulted or the existing results should be reviewed by unbiased domain experts.

In order to compare the results with the reference sets, it is necessary to identify similar functionalities and data objects (cf. Sec.6.9) that appear in each of them. We tried to avoid any biased fitting between the available instances in the reference sets and the result of the approach. However, this cannot be entirely ruled out.
Another threat to the validity is the claim that *Microservice 3* consists of the microservices *Order* and *Product*. Functions and data object that appear in *Microservice 3* are regarded to be successfully identified if they are in either of the two reference microservices. Strictly speaking, only two of the four reference services are identified by the approach. Consequently, all functionalities and data objects assigned to *Microservice 3* would be false positives. This, however, distorts the result in the same negative way, since most of the functions and data objects were assigned correctly, but to the coarse-grained *Microservice 3*.

7.5.2. External Validity

In contrast, external validity describes whether the evaluation results can be generalized and applied to other contexts [36]. That is to say, the approach also produces good microservice candidates for systems that are not in the context of a supermarket chain like CoCoME.
During the development of the approach, no domain restrictions were wittingly enforced. All assumptions were made with the intention of being context-independent. Further, *CoCoME* is structured like a typical, distributed and component-based information system. It is therefore to be expected that the approach also produces adequate results for other information systems.
Besides all the precautions, it is still possible that a fitting between CoCoME and the developed approach has happened. It is therefore necessary to apply the approach to other systems in different context to validate it's generality.

8. Conclusion

In the following section the chapters are summarized. Then the results of the research questions are presented. Second to last, the elaborated approach is discussed. The presentation of the limitations and the future work finish the thesis.

8.1. Summary

The aim of this thesis was finding a formal way to extract microservices using clustering on control flow and data flow. In Chapter 1, the topic is motivated and the research questions are posed. In Chapter 2, the monolithic software architecture and the microservice architecture is introduced and shortly compared. Further, challenges and benefits of the microservice architecture are outlined. Also, this chapter introduces use cases and the *Business Process and Model Notation (BPMN)*, which is a graph oriented language to describe business processes. In Chapter 3 *CoCoME*, the running example used to evaluate the approach is presented. *CoCoME* is a community case study and has recently been used in other projects to evaluate the results of microservice identification approaches. The current state of the art is introduced in Chapter 4, which provides the formal answer to **RQ1.1**: Several promising approaches and attempts to extract microservices are identified. Most of them require either a vast amount of initial input in form of diagrams, log files, graphs etc. or they require non-trivial user interaction during the extraction process. Also, some approaches cannot be applied to greenfield applications, as they need existing source code or at least the change history of the software development process. However, a main objective was to identify microservice candidates without detailed know-how and manual effort which non of them can satisfy.
To answer **RQ1.2**, the actual approach to identify microservices based on business process models is introduced in chapter 5. The approach is divided in several steps, each step is explained in a distinct section. First, the BPMN models are specified. Since CoCoME's systems specifications are given in form of detailed use cases, a transformation method is introduced to transform use cases in BPMN models. In the following, strategies to extract the data flow and the control flow of the BPMN models are explained. Subsequently, a procedure is explained to visualize the data and control flow information as two separate weighted graphs that represent the dependencies between either data objects or activities. The next step introduces a clustering algorithm that uses a fitness function to determine the fitness of a selected partition. This algorithm is applied to the graphs in order to identify highly cohesive and loosely coupled activity and data object clusters. In the last steps of the approach, four different methods to match the activity and data clusters are presented, where each combined cluster corresponds to a possible microservice candidate. In chapter 6, the approach is applied to the running example. In chapter 7, a *GQM Plan*

to answer **RQ1.3** efficiently is introduced. This includes the introduction of a metric and the presentation of two reference sets. Further, the results of the previous chapter are compared to the reference sets using the metric.

8.2. Outcomes

RQ1: How to identify microservices based on the system specifications?

Outcome: To that end, an approach was elaborated that uses clustering on control flow and data flow extracted from BPMN models.

RQ1.1: Which is an appropriate strategy to decompose a system into microservices?

Outcome: The control flow and the data flow are used to identify two separate sets of clusters: Activity clusters based on control flow dependencies and data object clusters based on data flow dependencies. Afterwards, both sets of clusters are matched in order to generate highly cohesive and loosely coupled microservices.

RQ1.2: How to identify possible microservices without detailed knowledge and manual effort?

Outcome: The approach consists of nine steps. The system specifications are in form of BPMN 2.0 models and contain control flow and data flow information. The information is extracted and visualized as two weighted graphs. Subsequently, a generic algorithm identifies dense activity and data object cluster. Based on the data object access between activity and data object clusters, the clusters are finally matched. Still, the manual effort is not negligible as the approach is not yet implemented. However, no additional effort, know-how or user-interaction is necessary, as soon as the BPMN models are specified.

RQ1.3: What is the accuracy of the approach?

Outcome: The overall accuracy of the approach is satisfying. No false microservice was identified when applying the approach to CoCoME. Only one of the identified microservices is more coarse-grained compared to the reference sets. In addition, most of the identified functionalities and data objects were allocated to the right service.

8.3. Discussion

Using use cases as input causes some trouble as the transformation into BPMN models with data needs is not trivial. It rather requires manual effort to detect synonymous tasks and data objects among all use cases to prevent a distorted result. This opposes the main purpose of the approach which is to reduce the required manual work and effort. However, it would be very difficult to create an approach that is capable of handling every form of input. Consequently, the first step, which is specifying the BPMN models, should rather be considered as prerequisite of the approach instead of being part of it. In case the system specifications are given as BPMN models with accompanying data reads and writes, the approach gets by on a minimum amount of required expertise and effort.
Section 5.2.3 introduces the creation of a graph based on the control flow information. Regarding possible alternatives in the control flow (XOR gateways), it was decided not to assess different choices in terms of different weighting. Yet, it is debatable whether this does not distort the actual control flow dependencies and consequently the results. Since the goal was not to identify perfect and fixed microservices, but to deliver candidates with little additional knowledge, this point is rather neglectable.
Another debatable topic is the equal weighting of all edges between activities. One can argue that some activities are more relevant and executed more often. For instance, the *Sale* process is probably performed more frequently than the creation a new product. Hence, activities covering the *Sale* process need to be connected using edges with higher weights to raise the probability of ending in the same service. However, this would require domain experts to judge the relevance of each activity, leading to a much more complex and user-interactive process.
Probably the most controversial issue is the identification of data flow dependencies (cf. Sec.5.2.5). As BPMN 2.0 is not capable to model the data flow but rather the data needs of the activities, it is necessary to extract data flow dependencies between two data object based on heuristics. However, these depend very much on the granularity of the BPMN models. The value for the parameter n needs to be adapted to the granularity, where fine-grained business processes demand for a higher value and vice versa. Additionally, the approach may not produce satisfying results if the granularity varies, that is if some business processes are much more fine or coarse grained than others. Consequently, it is perhaps necessary to think about additional input: The BPMN models are still used to extract the control flow information and to create activity clusters. In regard to data clusters, it might be necessary to use actual data flow diagrams that represent the exact data flow dependencies in order to create the data object related graph. Although this increases the necessary amount of input, it might improve the quality of the identified microservices. It is worth mentioning that the approach itself only needs to be adapted at two positions: i) *Extract Data Flow* and ii) *Create a weighted Graph using Data Flow*, both adaptions would be trivial steps if the input is expanded by data flow diagrams.
The evaluation also demonstrated how close the result is compared to an actual implementation. However, it shows how important a complete system specification is: The use case specification of CoCoME does not mention tasks like *Create Store* or *Create Products*. Consequently, the result does not contain these either. Since the approach requires no further human intervention, the input must be very precise and detailed.

The overall results are satisfactory. The identified microservices, including their functionalities and data objects are mostly correctly identified. Compared to the manual process as presented in Sec.7.2.2 which took days, the approach presented requires only a few hours.

8.4. Limitations and Future Work

Despite the overall satisfying results, the approach has its limitations. First of all, system requirements need to be transformed into BPMN 2.0 models with additional data needs. Even though the BPMN language is well known to describe business processes, the extension with the data objects is rather unusual. Hence, specifying those models might be a new task for the system designers.
In addition, it needs to be verified whether the approach is capable to detect different granular microservices. For instance, small services like *Order* and *Product* are not identified by the approach, because it merges them to a more coarse-grained microservice which has approximately the size of the other identified services. In this case, further research on different clustering algorithms needs to be conducted. Also, the possibility to choose the degree of granularity when applying the approach would be desirable. The current algorithm is not capable of finding various decompositions of a graph that vary in size.
In addition, we presented a conceptual algorithm to match cluster by using merging but also splitting. This needs further attention as it might solve the issue of different granularity of microservices. Above all, the currently used black box cluster matching process can lead to very large microservices, since only merging is used. Therefore, the conceptional white box clustering algorithm should be further investigated.
As already discussed in the previous section, the granularity of the BPMN models determines the choice of the parameter n, which influences possible data object dependencies. Apart from the fact that a wrong choice of the parameter distorts the result, it is questionable to apply the approach to BPMN models of different granularity. In other words, the approach is able to handle either coarse-grained or fine-grained BPMN models, but not a mix of both.
The microservice candidates generated by the approach are not fix and ready to implement but should rather used be as a recommendation to inspire the system architect. Additionally, microservices need a clear and well-defined external interface to provide shared functionality and data access to other services. This is not yet covered by the approach, although the information is already provided implicitly.

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] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A software architect’s perspective*. Addison-Wesley Professional, 2015.

[7] 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.

[8] 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.

[9] Niko Benkler. *Microservice-based implementation of CoCoME*. Accessed on 19.03.2019. URL: https://github.com/cocome-community-case-study/cocome-cloud-jee-microservices-rest.

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

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

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

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

[20] 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.

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

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

[23] 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.

[24] 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.

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

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

[27] 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.

[28] 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.

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

[30] 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.

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

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

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

[34] 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.

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

[36] Sarah Ransdell. “Educational software evaluation research: Balancing internal, external, and ecological validity”. In: *Behavior Research Methods, Instruments, & Computers* 25.2 (June 1993), pp. 228–232. ISSN: 1532-5970. DOI: 10.3758/BF03204502. URL: https://doi.org/10.3758/BF03204502.

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

[38] 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.

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

[40] *Types and Uses of Evaluation*.

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

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

[43] 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

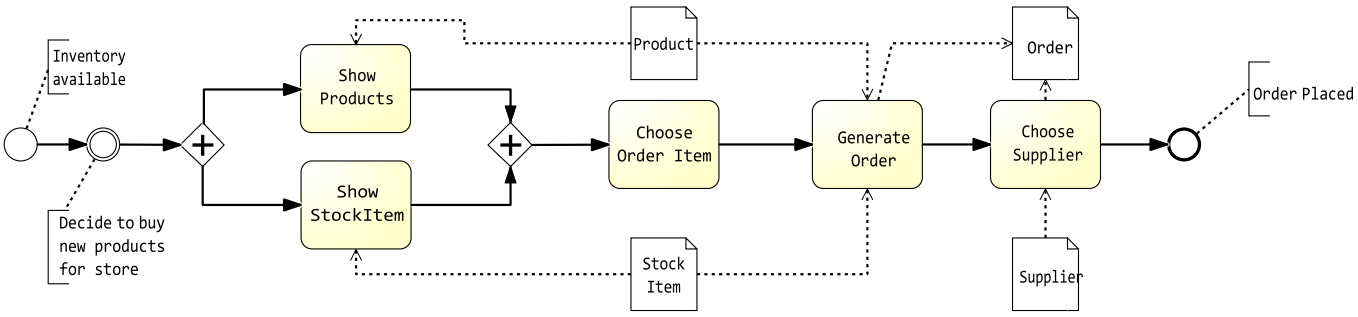

Figure A.1.: UC3 - Order Products

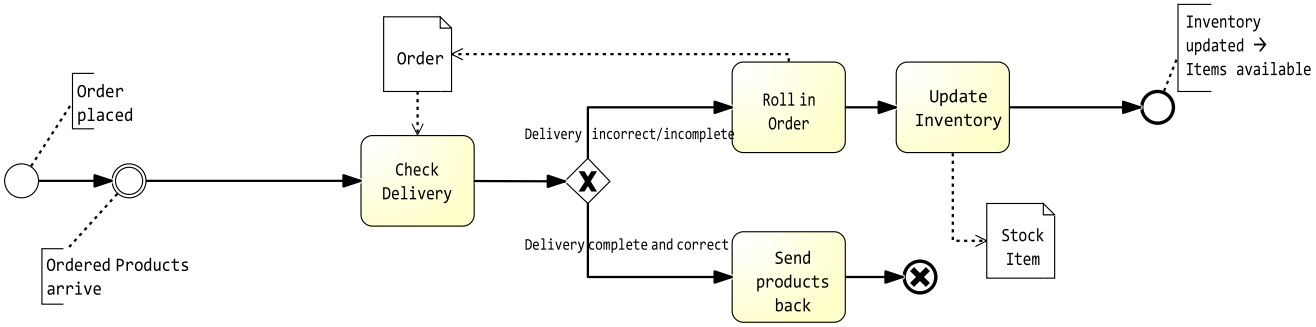

Figure A.2.: UC4 - Receive Ordered Products

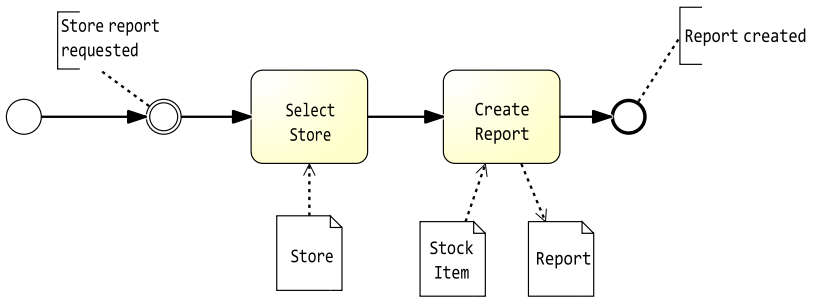

Figure A.3.: UC5 - Show Stock Reports

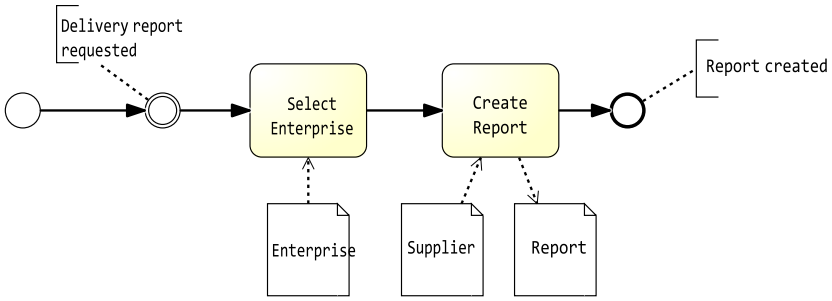

Figure A.4.: UC6 - Show Delivery Reports

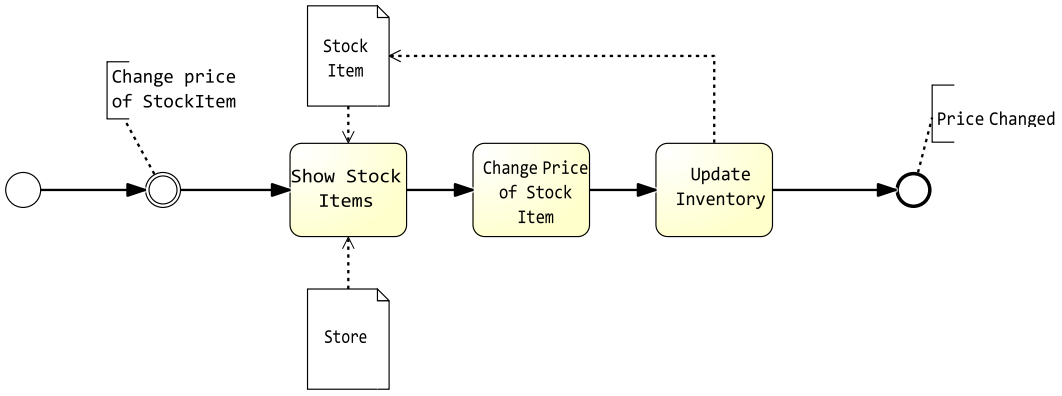

Figure A.5.: UC7 - Change Price

A.2. Control Flow Diagrams

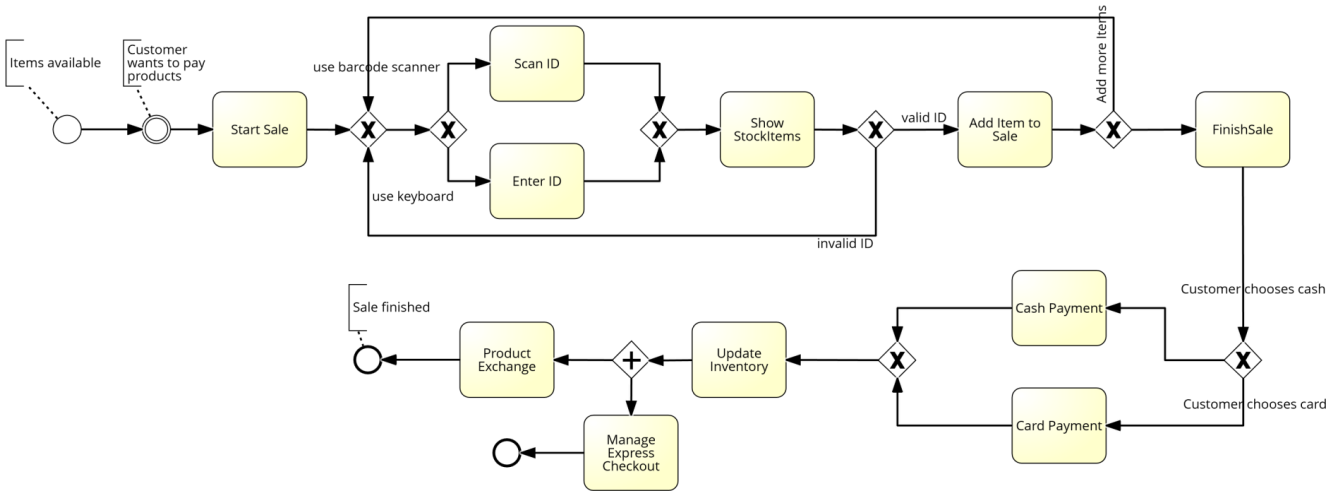

Figure A.6.: Control Flow UC1 - Start Sale

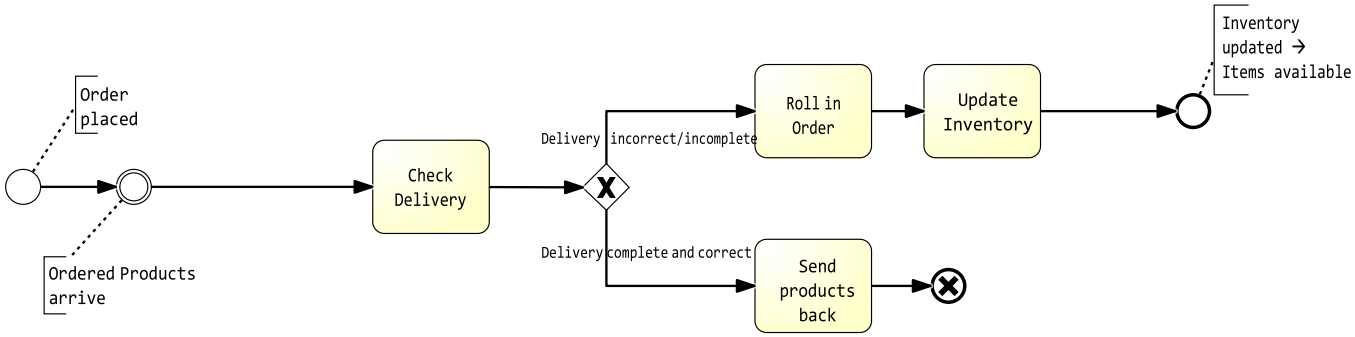

Figure A.7.: Control Flow UC4 - Receive Ordered Products

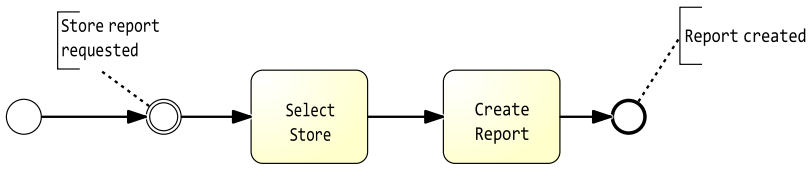

Figure A.8.: Control Flow UC5 - Show Stock Reports

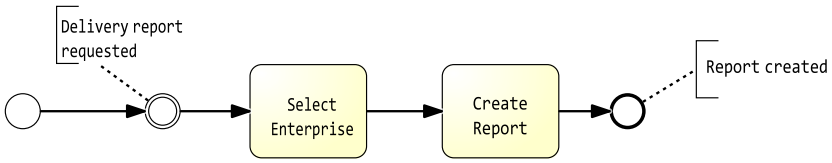

Figure A.9.: Control Flow UC6 - Show Delivery Reports

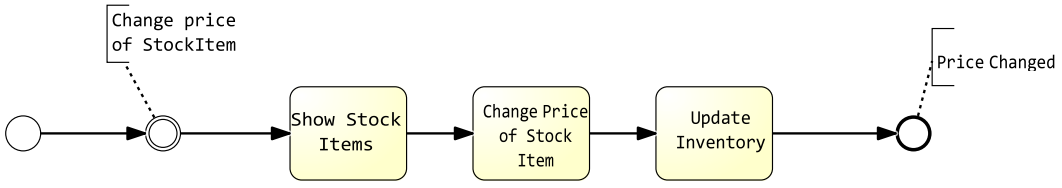

Figure A.10.: Control Flow UC7 - Change Price

A.3. Data Flow Diagrams

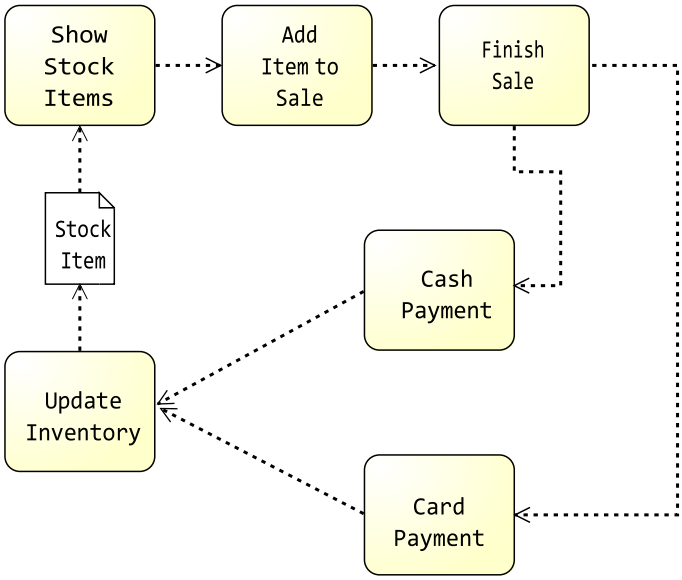

Figure A.11.: Data Flow UC1 - Start Sale

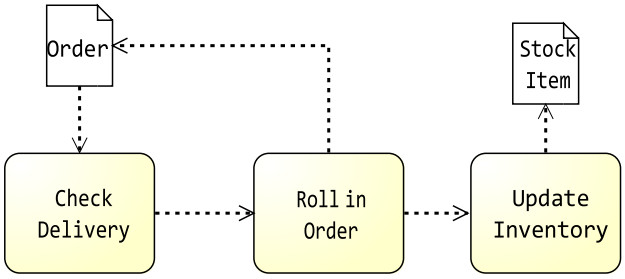

Figure A.12.: Data Flow UC4 - Receive Ordered Products

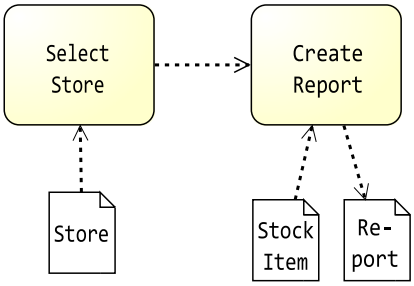

Figure A.13.: Data Flow UC5 - Show Stock Reports

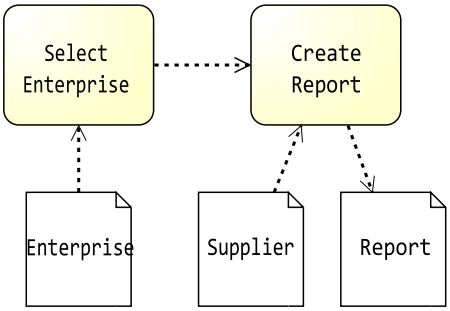

Figure A.14.: Data Flow UC6 - Show Delivery Reports

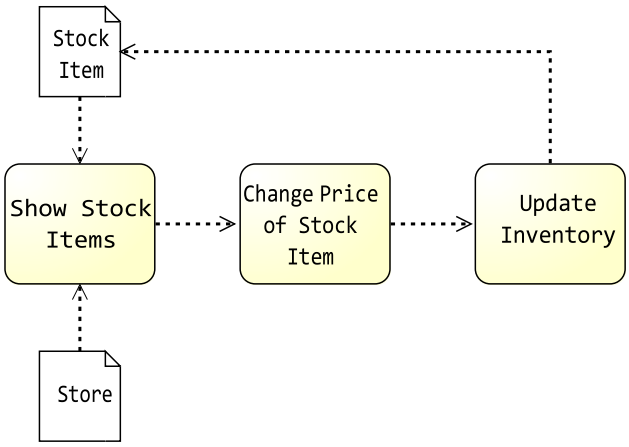

Figure A.15.: Data Flow UC7 - Change Price