MASTER

Artifact-centric log extraction and process discovery

Lu, X.

Award date:
2013

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
Artifact-Centric Log Extraction and Process Discovery

*Master Thesis*

Xixi Lu

**Supervisors:**
- Dr. ir. Dirk Fahland
- ir. Marijn Nagelkerke (KPMG)
- ir. Dennis van de Wiel (KPMG)
- Dr. ir. Irene Vanderfeesten

**Final version**

Eindhoven, September 2013
Abstract

The omnipresence of using Enterprise Resource Planning (ERP) systems to support business processes has enabled recording a great amount of (relational) data which indicates the behaviors of these processes. However, to allow classic process mining techniques to be applied on this data to discover process models and gain new insight, approaches have been proposed to convert this data into an event log, which is generally required by process mining techniques. These traditional conversion approaches tend to assume an isolated process with a clear notion of a case and a unique case identifier. This assumption has caused issues such as the known Data Convergence and Divergence problems in an ERP system setting, due to the fact that these processes comprise the life-cycles of various interrelated data objects, also known as artifacts, instead of a single object. In this thesis, a new semi-automatic artifact-oriented approach is presented which is extended from the XTract approach proposed by E.H.J.Nooijen et al. [21][22]. The new approach identifies various artifacts and discovers the life-cycle of each of these artifacts and their interrelated relations both on the artifact type level and on the event type level. An artifact-centric model in the proclet language [4] and diverse statistics for business analyses are obtained by using our approach. The presented approach is implemented and evaluated on multiple processes of ERP systems through case studies.

Keywords: Process Mining, Artifact-Centric Process Discovery, Relational Data to Event logs Conversion, ERP Systems
Preface

This master thesis is the result of my graduation project which completes my Business Information Systems study at Eindhoven University of Technology. The project was conducted in cooperation with KPMG IT Advisory and the Architecture of Information Systems group of the Mathematics and Computer Science department of Eindhoven University of Technology.

First of all I would like to thank my graduation supervisor, Dirk Fahland, for his guidance during my master project. I would like to thank Dirk for the weekly discussions, extensive reviews, and the fast and constructive feedback. Furthermore my thanks go out to Irene Vanderfeesten for being my reviewer.

Secondly, I would like to thank KPMG IT Advisory N.V. for allowing me to join the team. KPMG has given me the opportunity to work with experts in the field of data analytics, SAP, and Oracle and freely use the available resources in the organization. My colleagues have made my internship a really valuable experience where I learned to think about problems in a business perspective and where we had a lot of fun. Special gratitude goes to Marijn Nagelkerke and Dennis van de Wiel, my supervisors, who have helped me to apply my research in practice and conduct case studies. Moreover, I would like to thank Ruud who has provided me the opportunity and helped me conducting the case study.

I would like to thank Boudewijn van Dongen, Mike Bussel, and Rob Reijnders, without who I would not have come into contact with KPMG and thus would not have had the opportunity to do this project in the first place. I would especially like to thank Boudewijn who has been a great mentor and helped me to get started with this master project. I would like to express my gratitude to Erik Nooijen, whose thesis forms the fundamentals of this project.

Finally, I would like to thank my mother, my friends and my class mates. Without their support, I would never have come this far. This project concludes my five years study at Eindhoven University of Technology, during which I have met so many people such as Maikel, Han, Paul, Pim, Janôt, Quirijn, Sander, Alex, Qian, Qinrui, Xiyuan, Ruben, Thomas, Kosmas, Tatiana, Twan, Zheyi and many others who have helped me grow so much. Thank you.
Contents

Abstract iii
Preface v

1 Introduction 1
1.1 Thesis Context 1
1.2 Research Problem 7
1.3 Research Scope 9
1.4 Outline - Artifact Centric Approach 10

2 Preliminaries 13
2.1 Process Mining 13
  2.1.1 Event logs and the XES Event Log Format 14
  2.1.2 Process Discovery and ProM framework 16
  2.1.3 Process Models and Artifact-Centric Model 17
  2.1.4 Relational Databases and ERP systems 17
2.2 Literature Study - Traditional Log Extraction Approaches 18
2.3 Literature Study - Artifact-Centric Log Extraction Approach 20
  2.3.1 Open Issues of XTract 21

3 Artifact Type Identification 27
3.1 Data Schema Identification 28
3.2 Artifact Schema Identification 30
3.3 Artifact Identification 32

4 Interaction Identification 37
4.1 Artifact Type Level and Instance Level Interactions 39
  4.1.1 Direct Artifact Type level and Instance Level Interactions 39
  4.1.2 Indirect Artifact level Interactions 41
  4.1.3 Possibility of Identifying Event Level Interactions 44
4.2 Type Level Interaction Identification 44
  4.2.1 Artifact Interaction Graph and Direct Interactions 45
  4.2.2 Direct to Indirect Interactions 46
4.3 Mapping Creation and Log Extraction 48

vii
# CONTENTS

5 Artifact-Centric Process Discovery and Analyses ................................................. 51
  5.1 Logs and Life-cycle Discovery ................................................................. 52
    5.1.1 Event logs with trace level interactions ........................................... 52
    5.1.2 Life-cycle Miners .............................................................................. 54
  5.2 Event Type Level Interaction Identification .................................................... 54
    5.2.1 Interaction Discovery by Merging Logs ................................................ 54
    5.2.2 Interaction Discovery by using Criteria ................................................ 56
    5.2.3 Limitations ......................................................................................... 58
  5.3 Artifact-Centric Model Discovery ................................................................. 58
    5.3.1 Proclet System Creation .................................................................... 59
    5.3.2 Simple representation ........................................................................ 59
  5.4 Artifact-Centric Process Analyses ................................................................. 60
    5.4.1 Traceability ......................................................................................... 60

6 Implementation ................................................................................................. 63
  6.1 XTract2 ....................................................................................................... 63
  6.2 InteractionMiner ......................................................................................... 66

7 Case Studies ..................................................................................................... 69
  7.1 Case I - SAP Order To Cash Process .......................................................... 69
    7.1.1 SAP OTC process and data structure .................................................... 69
    7.1.2 SAP OTC - Extraction and Discovery .................................................. 70
    7.1.3 SAP OTC - Process Analyses and Discussion ....................................... 75
  7.2 Case II - Oracle Project Administration Process .......................................... 78
    7.2.1 Oracle PA Process and Data source ...................................................... 78
    7.2.2 Oracle PA - Extraction and Discovery .................................................. 79
    7.2.3 Process Analyses Result and Discussion ............................................... 82

8 Conclusion ....................................................................................................... 89
  8.1 Limitations and Future Work ....................................................................... 90

Bibliography ....................................................................................................... 92

A OTC Example .................................................................................................. 97

B Mapping Creation and Log Extraction ............................................................. 99

C Case Studies .................................................................................................. 101
  C.1 Case I - SAP OTC process ....................................................................... 101
List of Figures

1.1 The tables of the simplified OTC example ........................................ 2
1.2 A time-line regarding the creation of documents of the OTC example .... 3
1.3 An conceptual event log of the OTC example ................................. 4
1.4 A simple sequential causal graph of Sales Order S1 ........................... 5
1.5 A simple sequential causal graph of the OTC example ....................... 5
1.6 An proclet system of the OTC example regarding the creation event type 7
1.7 A simple representation of the proclet system of the OTC example ...... 7
1.8 The research scope of the thesis .................................................... 9
1.9 The research outline of this thesis ................................................ 11
2.1 Process Mining scope ............................................................... 14
2.2 Event logs structure ................................................................. 15
2.3 XES meta model ........................................................................ 16
2.4 The overall method of the original XTract ....................................... 21
2.5 The foreign key of OTC example that is identified automatically ........ 22
2.6 The artifact Sales Documents ...................................................... 24
3.1 Method Artifact Identification ...................................................... 28
3.2 The counterexample OTC for references ....................................... 29
3.3 Comparing the artifact schemas obtained using the XTract approach and our approach with respect to tables $T$ and the main table $T_m$ .... 32
3.4 Comparing the artifacts obtained using the XTract approach and our approach .................................................. 34
3.5 Artifact Type Identification ......................................................... 36
4.1 Interactions levels ........................................................................ 38
4.2 Method Interaction Identification .................................................. 39
4.3 The artifact type level interactions of the example. .......................... 41
4.4 Indirect Interactions .................................................................... 42
4.5 Type level interactions ................................................................. 44
4.6 Mapping Creation and Log Extraction .......................................... 50
5.1 Method Artifact-Centric Process Discovery and Analysis .............. 52
5.2 An artifact centric model of the OTC example ................................ 62
6.1 Architecture of XTract2 ............................................................. 64
6.2 Architecture of InteractionMiner ................................................ 64
6.3 XTract2 - (A1) Import Data schema ............................................. 64
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4</td>
<td>XTract2 - (A2) Artifact schema identification and (A3) Artifact identification</td>
</tr>
<tr>
<td>6.5</td>
<td>XTract2 - (B) Interaction Identification and Interaction Graph</td>
</tr>
<tr>
<td>6.6</td>
<td>XTract2 - (C) Log extraction : mapping with interactions and an example of event log with interactions</td>
</tr>
<tr>
<td>6.7</td>
<td>Selecting event logs for InteractionMiner</td>
</tr>
<tr>
<td>6.8</td>
<td>The dialog to select the life-cycle miner and the merged log miner</td>
</tr>
<tr>
<td>6.9</td>
<td>The proclet system view of the artifact model in the visualizer</td>
</tr>
<tr>
<td>6.10</td>
<td>The simple representation view in the visualizer</td>
</tr>
<tr>
<td>7.1</td>
<td>SAP OTC process - relational structure</td>
</tr>
<tr>
<td>7.2</td>
<td>SAP OTC process - tables and record counts</td>
</tr>
<tr>
<td>7.3</td>
<td>SAP OTC process - sales order lifecycle including the changes of lines</td>
</tr>
<tr>
<td>7.4</td>
<td>SAP OTC process - artifacts</td>
</tr>
<tr>
<td>7.5</td>
<td>SAP OTC process - interaction graph</td>
</tr>
<tr>
<td>7.6</td>
<td>SAP OTC process - artifacts with type level interactions extracted</td>
</tr>
<tr>
<td>7.7</td>
<td>SAP OTC process - artifact-centric model with simple representation</td>
</tr>
<tr>
<td>7.8</td>
<td>SAP OTC process - the Orders artifact with change event types</td>
</tr>
<tr>
<td>7.9</td>
<td>SAP OTC process - artifact-centric model with outliers</td>
</tr>
<tr>
<td>7.10</td>
<td>SAP OTC process - artifact-centric model with the discovered interactions</td>
</tr>
<tr>
<td>7.11</td>
<td>SAP OTC process - artifact-centric model with interactions</td>
</tr>
<tr>
<td>7.12</td>
<td>SAP OTC process - artifact-centric model with outliers</td>
</tr>
<tr>
<td>7.13</td>
<td>Oracle PA process table record counts</td>
</tr>
<tr>
<td>7.14</td>
<td>Interaction graph of PA artifacts</td>
</tr>
<tr>
<td>7.15</td>
<td>The artifacts created for Oracle PA process</td>
</tr>
<tr>
<td>7.16</td>
<td>Three event logs extracted</td>
</tr>
<tr>
<td>7.17</td>
<td>A proclet system discovered using merging logs method</td>
</tr>
<tr>
<td>7.18</td>
<td>A proclet system discovered by using the max number of event level interactions criterion</td>
</tr>
<tr>
<td>7.19</td>
<td>A proclet system discovered by using the existing precedence criterion</td>
</tr>
<tr>
<td>7.20</td>
<td>A proclet system discovered by using the existing precedence criterion</td>
</tr>
<tr>
<td>7.21</td>
<td>A proclet system discovered by using the existing precedence criterion</td>
</tr>
<tr>
<td>7.22</td>
<td>Expenditures created after projects closed</td>
</tr>
<tr>
<td>7.23</td>
<td>Expenditures created after projects closed in database</td>
</tr>
<tr>
<td>7.24</td>
<td>Four projects of which the closed date is more than one day earlier than its last update</td>
</tr>
<tr>
<td>7.25</td>
<td>Eight artifacts obtained from expenditure items</td>
</tr>
<tr>
<td>7.26</td>
<td>Interactions between the eight expenditure artifacts and the project life cycle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>Data schema of the OTC example</td>
</tr>
<tr>
<td>B.1</td>
<td>Mapping</td>
</tr>
<tr>
<td>B.2</td>
<td>Mapping data structure</td>
</tr>
<tr>
<td>C.1</td>
<td>SAP OTC tables - primary keys</td>
</tr>
<tr>
<td>C.2</td>
<td>SAP OTC tables - foreign keys</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Reference keys .................................................. 30
3.2 An example of the Sales Order artifact .......................... 33

4.1 Direct interactions \(A_{\text{SalesOrder}}, F_2, A_{\text{ReturnOrder}}\) instances .......................... 41
4.2 Direct interactions \(A_{\text{ReturnOrder}}, F_2, A_{\text{ReturnDelivery}}\) instances .......................... 41
Listings

4.1 Interaction Count Selection Query ........................................... 40
4.2 Interaction Instances Selection Query ..................................... 40
7.1 A query verifying payments received date earlier than invoice placed in AR .... 76
Chapter 1

Introduction

This master thesis, completed as part of the Business Information System master at Eindhoven University of Technology, is carried out within the KPMG Advisory N.V. (KPMG)\(^1\) and the Architecture of Information Systems (AIS) group of the Mathematics and Computer Science department of TU/e.

In this chapter, we start with introducing the current state of business process analyses and explaining the main difficulties that obstruct process mining techniques from being applied within business information systems using an example in Section 1.1. Then, we list the current approaches found in the literature that address these difficulties and discuss the issues found related to these approaches. Next, we present the artifact-centric approach found in the literature that addresses these issues. We shortly motivate why the artifact-centric approach is more suitable, followed by the open issues related to the current artifact-centric approach, which form the basis of our research problem defined in Section 1.2 and the research scope defined in Section 1.3. An outline of our artifact centric approach and of this thesis is given in Section 1.4.

1.1 Thesis Context

The omnipresence of using information systems, such as Enterprise Resource Planning (ERP) systems, Work flow management systems (WfMS), Customer Relation Management (CRM) systems, within corporations to support business processes has enabled the recording of a great amount of digital data describing the tasks done for the cases handled by corporations. Analyzing this data to retrieve useful information to gain insight into the occurrences in reality and to improve processes is currently a major topic in information technology. A new emerging type of data analysis techniques known as Process Mining\(^3\) has proven its usefulness by offering a new insight into business processes. The focus of these techniques is analyzing and visualizing continuous, concurrent data. One of the goals of process mining techniques is to discover a process model when given an event log, denoted as process discovery. As process discovery techniques mature, business users also tend to use the discovered model exploratively to detect deviating flows in business processes since the models obtained illustrate the real executions of processes rather than wishful thinking documented in a hand-made model.

\(^1\)http://www.kpmg.com/nl/en/services/advisory/pages/default.aspx
Figure 1.1: The tables of the simplified OTC example

Unfortunately, an event log, generally required by process discovery techniques, has a process oriented data structure which has very different characteristics than the data (or document) oriented structure of relational databases that are widely used by existing information systems. To be able to apply process mining techniques within data-centric information systems, event logs have to be extracted from the relational data stored in these systems.

To illustrate the difference between a relational data source and an event log, we introduce a simple example of the Order to Cash (OTC) process commonly supported by ERP systems such as SAP\(^2\) and Oracle\(^3\). In short, the OTC process starts with customers placing orders. Then, organizations fulfill the orders by delivering the goods and sending invoices to bill the cost and receive payments from customers. Organizations use the ERP system to store these sales orders, deliveries, invoices and payments that are related to the OTC process in tables. An example of the relational data is shown in Figure 1.1 and explained in detail in Appendix A. This OTC example will be used throughout this thesis to demonstrate various issues, concepts, definitions and methods. We briefly explain the process executions that have led to the data in Figure 1.1, but only related to the creation of documents for the sake of brevity. A time-line of the execution is shown in Figure 1.2.

\(^2\)http://en.wikipedia.org/wiki/SAP_ERP

\(^3\)http://en.wikipedia.org/wiki/Oracle_E-Business_Suite
in which a distinction is made between the creation of documents that is related to the sales order S1 (below the line in Figure 1.2) or to S2 (above the line in Figure 1.2). The process starts with a customer placing a sales order S1, which is created in the system on 16-5-2020. Then a partial delivery D1 is done on 18-5-2020, and the invoice B1 is created two days later. On 22-5-2020, another part of the sales order S1 is delivered according to the delivery document D2 which is invoiced with document B2 on 24-5-2020. On 17-5-2020, the same customer places another sales order S2, which is also invoiced within the same billing document B2. However, the delivery D3 related to the sales order S2 is executed after the billing document B2 on 25-5-2020. Days later, a return order S3 is placed for the sales order S1 and return delivery D4 is executed.

![Timeline](image)

Figure 1.2: A time-line regarding the creation of documents of the OTC example

Using this example, we can show the differences between an event log structure and a relational data structure. A main difference is caused by an assumption made by process mining techniques: one process execution has a clear notion of one case (e.g., a sales order), and the executions of the tasks on this case in this process result in a trace of events recorded for this case in the event log for this process. The events of a case are ordered according to their timestamps. For the OTC example, if we consider the sales orders as the cases, we can relate the deliveries and invoices to the two sales orders S1 and S2, which result in two traces of events constituting the event log shown in Figure 1.3. The relational data source shown in Figure 1.1 and the conceptual event log shown in Figure 1.3 illustrate clear differences in structure. The event log shown in Figure 1.3 has a clear notion of a case, i.e., Sales Orders. Each trace in the event log is related to one sales order, and all events in the trace are related to the sales order and compared with respect to the sales order. The relations between the events, e.g., the relations between the deliveries and invoices, are omitted. In contrast, the documents stored in the database have varying definitions and structure (i.e., different table and columns), and the records can be related to various other records, e.g., sales orders, deliveries, invoices. Moreover, the data related to a process or process instances might be
spread over the tables. Thus, the relational data related to the execution of a process has to be collected and converted to an event log. A conversion definition between a relational database of relational data and an event log, also known as a (log) mapping, is a specification which indicates specific part of the relational data should be converted to a specific part in the event log.

During the literature study, various approaches have been found to support the conversion from relational data to event logs. The goal of most approaches, proposed by several studies such as M. van Giessels [13], I.E.A. Segers [26], J. Buijs [9], D. Piessens [24], and A. Roest [25], is to extract one event log which describes a single, isolated process. We categorize these approach as the traditional log extraction approaches (also refer to as the traditional approaches). Many issues have been found regarding the traditional log extraction approaches. For instance, no clear definition of a case could be found in a data centric system since there are many different definitions of cases such as sales orders, deliveries and invoices identified in the OTC example. This lack of a clear notion of a case has lead to two well known problems: data divergence and data convergence.

The data divergence problem is defined as the situation when a case is related to multiple events of the same event type. Figure 1.3 shows that the case sales order S1 has two Delivery Created events D1 and D2 and two Invoice Created events B1 and B2. If we draw a simple causality net by only using the trace S1, we obtain the model shown in Figure 1.4. Business users immediately notice the edge from Invoice to Delivery and find this edge strange as they think the edge indicates that there are invoices created before the related deliveries. However, this edge actually means that there is an invoice B1 created before a delivery D2, both of which are related to the sales order S1 but not related to each other. This complexity and ambiguity of the process model increases when more deliveries and invoices are linked to the case, as the divergence problem also introduces loops. Now, if we include the trace S2, a similar model shown in Figure 1.5 is obtained with the same strange edge from Invoice Created to Delivery Created. However, this time there really is an
CHAPTER 1. INTRODUCTION

Figure 1.4: A simple sequential causal graph of Sales Order S1.

Figure 1.5: A simple sequential causal graph of the OTC example.

invoice B2 created before its related delivery document D3, which is an outlier and might indicate risks or faulty configurations in the process. Identifying these strange but true positive sequential dependencies between related invoices and deliveries is the aim of business users, which is disturbed by the divergence problem. Solving divergence should eliminate the false positives.

The problem of data convergence is defined as the situation when one event is related to multiple cases. For example, when considering the sales orders as the notion of a case and the creation of invoices as events, the Invoice Created event of the invoice B2, which is related to two different sales orders S1 and S2, will be extracted twice, as illustrated by the event log in Figure 1.3. Traditional process mining techniques consider the event Invoice Created B2 as two different events. Together with the creation of invoice B1, we obtain three Invoice Created events as shown in Figure 1.5, whereas there are only two invoices B1 and B2. Thus, the data convergence problem leads to extracting duplicate events and biased statistics.

Choosing different notions of a case for the process definition is proposed in different literature as a solution to the divergence and convergence problem. However, this solution might solve the divergence and convergence partially but not completely. Take the OTC example, choosing the invoices as the case definition, the many-to-many relation between the invoices and sales orders causes the event log obtained to still suffers from divergence and convergence. Choosing the deliveries as case definition solves the divergence problem, but worsens the convergence problem, e.g. also the sales order created S1 is extracted as an event for D1 and D2. Furthermore, it is also very difficult for the traditional approaches to define or to retrieve an optimal definition of a case from all possible case
definitions found in relational data. As ERP systems store various documents related to a business process execution, and the relational structure allows each document to have its own execution which may or may not be related to other documents, business processes supported by such an ERP system can be generally viewed as a set of documents and each has its own lifecycle. Moreover, as ERP systems support the operation of an entire organization which has dozens large processes, it is almost impossible to describe the behavior of a system using only one process definition. These issues have led to another type of log extraction approach and process mining techniques: the artifact-centric approach.

Artifacts are business entities described by both an information model and a lifecycle. For the OTC example, we can consider sales orders, deliveries and invoices as different artifacts, each has its own life cycle and interactions with each other. The relations between artifacts can be viewed as interactions between artifacts during their lifecycles, in which the data divergence and data convergence between artifacts are allowed. For example, the creation of sales orders might lead to the creation of multiple deliveries. This concept of artifacts is first introduced by A. Nigam and N. Caswell [20] and put into practice by D. Cohn and R. Hull [10]. An artifact-centric process model describes a process as multiple collaborative artifacts, each with its own life cycle and interactions with each other. A formal notation for an artifact-centric model is the proclet system proposed by W. van der Aalst et al. [4], which will be explained in Section 2.1.3. An example of a proclet system of the OTC example is shown in Figure 1.6. For each document type found in the OTC example, we consider it as an artifact (type) with its own life-cycle, illustrated by the large gray rectangle. The interactions are represented by the edges between the artifacts. Since the proclet system might be difficult to be explained to business users, a more abstract representation like Figure 1.7 might be more suitable.

Traditionally, users conducted interviews to manually construct artifact-centric process models for business process management. However, this approach is very time consuming, since an organization might have thousands of artifacts. E. Nooijen et al. [21][22] were the first and the only one who proposed an automatic approach to identify artifacts from a given relational data and extract artifacts as event logs for applying process mining techniques. E. Nooijen has implemented the approach as a tool named XTract.

Based on the result of literature study, we believe that the artifact-centric approach is a more suitable way for conducting process mining within data-centric systems such as ERP systems. First, the notion of interactions in an artifact-centric model allows the model to express one-to-many and many-to-one causalities between cases, which can solve the data divergence and convergence problems and obtain a simpler process model. Moreover, the notion of artifacts allows us to decompose a large process into smaller collaborative processes (i.e. life-cycles) of different cases. This decomposition further decreases the complexity of individual life-cycles and improves the scalability of applying process mining within data-centric systems. Furthermore, the artifact-centric approach is actually a generalization of the traditional log extraction approach because we can also consider the large process as one artifact and still obtain the same complex life-cycle of this artifact. For example, considering the sales orders of the OTC example as one artifact, and the creation events of deliveries, invoices, return order and return delivery constitute the sales orders’ life cycle, we obtain the same life-cycle as the process model shown in Figure 1.5.

Compared to the single case definition view of traditional log extraction approaches, the notion of artifacts gives a “conceptual lens” that allows us to interpret relational data differently. Unfortunately, the original XTract approach (or the tool) introduced by E. Nooijen et al. [21][22] is still
CHAPTER 1. INTRODUCTION

incomplete indicating a low practical value. We list here some issues and will discuss them in detail in Section 2.3. First, no interactions between artifacts were identified nor extracted by the original XTract approach, which left the artifacts’ life cycles as single, isolated processes. This incompleteness indicates that no complete artifact-centric model can be created (semi-) automatically yet from a given data source. Moreover, the XTract approach was unable to identify multiple artifacts sharing the main table: the XTract can only consider the Sales Documents (SD) table of the OTC example as one artifact and is unable to distinguish the artifacts Sales Order and Return Order based on the document types from the SD table. The XTract tool also does not support any changes of artifacts identified. In addition, the XTract approach has not addressed the data convergence and data divergence explicitly.

As it is useful to apply process mining techniques within ERP systems, and the artifact-centric approach is a more intuitive way to perform this compared to the traditional log extraction approaches, the existing artifact-centric approach has to be extended for artifact identification, and new methods have to be developed for interaction identification and artifact-centric process model discovery.

1.2 Research Problem

In section 1.1, we have briefly introduce the thesis context and issues related to traditional log extraction approaches as well as the artifact-centric log extraction approach. In this section, we give the problem definition of this thesis and divide the problem into three sub problems.

We have argued that the artifact-centric approach is a more suitable way for conducting process mining within data-centric systems than the traditional log extraction approach. First, we can use the notion of interactions to express one-to-many and many-to-one causalities between artifacts,
solve the data divergence and convergence problems, and obtain a more intuitive model. Second, the notion of artifacts allows us to decompose a large process into smaller collaborative processes (i.e. life-cycles) of different cases and therefore improving the scalability of log extraction and process mining. Third, because we can also consider a large process also as one artifact, the artifact-centric approach is actually a generalization of the traditional log extraction approach.

We have also discussed the current artifact-centric approach is incomplete. For example, no method has been found in the literature which can identify interactions between different process definitions (i.e. between artifacts if in an artifact-centric context) or between cases or between events, neither on the database level nor on the event log level. Thus, no approach supports the (semi-) automatic discovery of an artifact-centric model yet, when given a data source. Due to the business context of this thesis, it also is important that the identified artifact-centric model can be used by business analysts to analyze processes. Therefore, we define the research problem as follows:

Given a relational data source, we would like to support business analysts to (semi-) automatically discover an artifact-centric model, which describes the process of several collaborative processes (or life cycles of artifacts), each of which has its own life cycle, and interactions with each other, and to be able to use this discovered model to perform process analyses.

We assume that the relational data source contains information about the documents in the process, their relations to each other and timestamps of the tasks performed on these documents that are relevant for the process analyses. An example of the relational data source is the OTC tables shown in Figure 1.1. In addition, if the data schema is too complex to be identified automatically, we assume that there is knowledge available of the data schema (i.e. primary keys and foreign keys).

Since many existing technologies provide partial solutions, we can divide the research problem into sub problems to use as many existing techniques as possible to (partially) solve the problem. Especially, XTract approach provides a prototype which allows users to connect to different types of relational databases and provides functions to identify simple, isolated artifacts and to create event logs. Consequently, each event log that is created based on an artifact can be used as input for existing process discovery techniques to obtain a process model (or a life cycle in an artifact-centric context). In addition, since the event log format has been standardized using the XES format, creating event logs with interactions will allow more artifact centric discovery techniques to be developed and applied independent of the method proposed in this thesis. The artifact-centric model obtained should have formal definitions to allow researchers to further investigate issues in an academic context. In addition, due to the business context of this thesis, the artifact-centric model that is shown to the analysts should be simple and understandable, and allow analysts to identify outliers. The analysts should be able to retrieve cases (or case identifiers) and statistic information to verify and assess the process. For aforementioned reasons, we divide the research problem into to three sub problems:

(I) Given a relational data source, we would like to support business analysts to (semi-) automatically extract event logs, where each event log describes the life cycle of an artifact as well as the interactions of this artifact with other artifacts.

(II) Given a set of event logs which also contains data of interactions with each other, we would like to discover a formal artifact-centric model including interactions among the life cycles.
(III) Given a formal artifact-centric model together with the corresponding set of event logs, we would like to be able to visualize a model that is intuitive for business users and to obtain information about outliers and statistics.

Due to the large scope and the interest of the business analysts involved in this thesis, the primary goal of this thesis is to be able to obtain an artifact-centric model that supports the business analysts performing process analyses (III). The usability of the GUI of the prototype is of secondary concern. Therefore, case studies are conducted to evaluate the artifact-centric models obtained instead of the usability of the tools.

1.3 Research Scope

We have defined the research problem. To solve the problem, we need an approach that supports analysts to extract event logs including the interactions between the logs, and an approach that discovers an artifact-centric model, outliers and statistics from these event logs. Furthermore, the model obtained is evaluated via case studies to improve the understandability. In this section, we define the research scope of this thesis.

![Figure 1.8: The research scope of the thesis](image)

A high level overview of the research scope (and the solution approach) is demonstrated in Figure 1.8. We briefly explain each element and the reason why it is included. First, the data sources obtained is generally derived from interactions between human users and information systems for business process executions. Since the recording of process executions is done by information systems, some knowledge of the information systems to be mined is desired. In this thesis, the knowledge of the relevant information systems is provided by the system experts at KPMG or obtained during the interviews with advisors.

The data sources recorded by information systems can be considered as input for the log extraction method. KPMG has extensive experience with data analysis and has developed a data analysis
platform (Facts2Value) that supports extracting, analyzing and reporting data from the client’s information systems for audit and advisory purposes. Currently, over two hundred Facts2Value analyses engagements are performed annually. Facts2Value is still being expanded and improved. Because the data downloaded by KPMG were uploaded to the data warehouse on Microsoft SQL servers to make them available for performing Business Intelligence analysis, we also used these data as input in this thesis to conduct case studies and test our artifact-centric process analysis approach.

Using the relational data sources as input, the log extraction method, denoted by the first black box (with a question mark), is the solution that supports analysts to extract event logs including the interactions between the logs (i.e. problem (I)). In addition, the log extraction method should support analysts to select, change or construct the artifacts (i.e. process definitions) and interactions desired. A list of event logs with interactions in XES format is then obtained and used for the discovery of an artifact-centric model. The mining method, denoted by the second black box, returns an artifact-centric model including interactions among the life cycles (which aims to address problem (II)). The artifact-centric model discovered should be intuitive and easy to understand.

It is important to include the human element, since the goal of process mining is to provide useful information for business analysts. Specifically, due to the business context of this thesis, the business advisors should be able to use our approach (i.e. the prototype built for support), and the model discovered should be easy to understand by business users. Therefore, case studies have been conducted with both business analysts and clients to evaluate the process models obtained (which aims to address problem (III)).

1.4 Outline - Artifact Centric Approach

In the previous sections, we have defined the research problem and scope to semi-automatically discover an artifact-centric model from a given data source. In this section, we outline the report by briefly introducing our artifact-centric approach for the thesis problem including sub methods to the sub problems. An overview of the approach is shown in Figure 1.9.

First, in Chapter 2, the preliminary concepts that are related to the research scope illustrated in Figure 1.8 are investigated by conducting literature study: event log format in Section 2.1.1, process discovery techniques (i.e. the “mining” element) in Section 2.1.2, process models in Section 2.1.3, information systems (also data sources) in Section 2.1.4, and the traditional log extraction methods in Section 2.2. In addition, the artifact-centric log extraction approach XTract is also investigated in detail in Section 2.3, since we use the XTract approach as a start point of our approach. We also discuss the open issues of the XTract approach in detail and introduce our solution to the issue by referring to the specific section describing the solution.

In Chapters 3 and 4, we present our artifact-centric approach to the log extraction problem (I). We divide the log extraction problem (I) into three phases: (A) artifact type level identification, (B) type level interaction identification, and (C) log extraction.

In the first phase (A) artifact type identification explained in Chapter 3, we would like to identify a set of stand-alone artifacts from the given data sources. Each artifact contains all data related to the life cycle of the artifact. We consider the artifact type identification as a separate phase since the artifacts obtained (without interactions) can already be extracted as event logs which can be used to identify the life cycles. Moreover, the existing XTract approach can be used and
extended to solve this sub problem. Our approach to identify artifact types consists of three steps: (A1) given data sources, we generalize the definition of foreign keys (named references) and allow analysts to import schema information to obtain the data schema in Section 3.1; (A2) given the data schema, we identify the artifact schemas to discover similar artifacts while explicitly addressing the convergence and divergence problems in Section 3.2; (A3) we allow identifying multiple artifacts from a given artifact schema, explained in Section 3.3.

In the second phase (B) type level interaction identification, we identify the (type level) interactions between artifacts. These interactions are necessary to relate different artifacts to each other and to investigate the causal relations between the life-cycles of artifacts, e.g. the Creation of the Sales Order artifact has led to the Creation of the Delivery artifact. Furthermore, interactions between artifacts can be used to express the one-to-many, many-to-one, and many-to-many relations, which solve the data divergence and convergence problem and help to create a more intuitive model. In this phase (B), we first define the interactions and investigate the possibility of extracting interactions on the database level in Section 4.1. Then, we propose our approach to identify (B1) direct and (B2) indirect interactions between artifacts in Section 4.2.

After identifying the (type level) interactions, the artifacts are complete. In the third phase (C) Log Extraction, we use the artifacts to create log mappings (i.e. conversion definitions) for extracting the relational data of each artifact (type) as an event log. This phase (C) is necessary to obtain the event logs, which can be imported into process mining tools to discover process models. It is logical to divide the interaction identification and the log extraction into two phases, since one
can skip the interaction identification if they are not interesting. Moreover, one can propose different methods for interaction identification or log extraction. In this thesis, the log extraction function provided by the original XTract approach is mainly reused with minor changes. Our extensions are explained in Section 4.3. One can implement new methods for writing logs since the existing functions show some performance issues. Both phases are illustrated in Chapter 4.

Our approach to address the mining problem, explained in Chapter 5, is necessary to obtain an artifact-centric process model from the set of event logs (with interactions). After solving the extraction problem, we obtain a set of event logs, each of which describes the life cycle of an artifact type (e.g. Sales Order), and each trace in this event log represents the life cycle of an artifact instance (e.g. S1). This set of event logs is used by our approach to discover an artifact-centric model and other information such as outliers and statistics, which addresses the second problem (II) and the third problem (III). We first show how to identify individual life-cycles of artifacts in Section 5.1. Then, we show two different methods to identify interactions between event types of different life-cycles in Section 5.2, in which we also illustrate how to identify unusual interactions. Finally, we create an artifact-centric model by using the life-cycles and the interactions between their event types, explained in Section 5.3. For business users, it is important to be able to retrieve cases from a given causal relation, and this is shown in Section 5.4.

In Chapter 6, the two implementations XTract2 and InteractionMiner for, respectively, the extraction and mining problem are explained in Sections 6.1 and 6.2. The architectures of the two implementations together with a set of screen shots explaining the functions of the tools are also shown.

Case studies have been conducted using the data provided by KPMG to evaluate our approach and whether the third problem (III) is addressed. The case studies were performed for two different processes of two different types of ERP systems, Oracle and SAP: the Order to Cash (OTC) process of SAP and the Project Administration (PA) process of Oracle. The SAP - OTC process was chosen by the company supervisors. The PA process was chosen by an available customer. The result of the case studies are included and discussed in Chapter 7.

Finally, this thesis is concluded by Chapter 8 in which a summary of the thesis is given, and limitations of the approach are discussed. We also include a list of future work to discuss the possibility to overcome the limitations and further improve the artifact-centric approach in general.

In the following chapters, we use the words ‘XTract approach’ when referring to the original XTract approach, whereas the words ‘our approach’ or ‘our method’ are used when referring to the method that is newly designed to solve the specific problem within the given context.
Chapter 2

Preliminaries

We have defined our research scope and goal in the previous chapter. Before going into the details of our approach, the preliminary concept of each component in the research scope shown in Figure 1.8, i.e. process mining, event logs, process models, ERP systems and log extraction approaches, are introduced in this chapter since these concepts will be used throughout this thesis. First, we briefly describe process mining and its scope in Section 2.1. Within Section 2.1, we further explain the components of the thesis scope which are also related to the process mining scope: (Section 2.1.1) event logs and the XES format, which is extracted from the data sources; (Section 2.1.2) process discovery techniques which are used to mining the event logs; (Section 2.1.3) the process models returned by the discovery techniques; (Section 2.1.4) the relational data sources and ERP systems used as the source of event log extraction. Finally, the traditional approaches extracting the event logs and the original artifact oriented XTract approach are discussed in Sections 2.2 and 2.3, respectively.

2.1 Process Mining

In this section, we briefly introduce the goal and scope of process mining as illustrated in Figure 2.1 and relate the elements of process mining to an artifact-centric context. Process mining is a set of techniques to "discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs" [3]. Note that the research scope of this thesis shown in Figure 1.8 is aligned with the process mining scope: the data sources component is similar to the information systems component. However, the log extraction is out of the process mining scope since traditional mining techniques assumed the existence of event logs. Therefore, the problem of log extraction was explicitly added to Figure 2.1 to show its position in the classic process mining scope and to help readers compare the classic process mining scope to our research scope. When applying process mining techniques on the event logs recorded from processes, insight into control flow dependencies, data usage, resource utilization and various performance related statistics of the processes can be obtained.
2.1.1 Event logs and the XES Event Log Format

Most process mining techniques require an event log as input. In this section, a general event log (structure) is first explained. Then, since we have introduced our artifact centric approach in Section 1.4, we would like to give a comparison between the terminologies used in the traditional process mining context and the terminologies in the artifact centric context on a conceptual level. Next, a formal definition of an event log used for reasoning is given. This section is concluded by introducing the event log format XES which is used in this thesis.

In general, an event log comprises lists of events. Each list of events is called a trace or a process instance and is recorded from the tasks executed on a certain case instance going through the process. A common structure of an event log, drawn by J. Buijs [9], is shown in Figure 2.2 on the right hand side. On the model level (or process definition level), a process specifies the activities that should be executed in a certain order. A process can be instantiated to the case instances going through this process, also called process instances. The event types, instantiated from the activities, are performed on the case instances of the process which are recoded as events on the log level containing the data such as the time when the event is executed (i.e. timestamp), the resource who executed the event, and other attributes. An ordered list of events performed on a case instance is called a trace.

Figure 2.2 on the left hand side compares the definitions of event logs in a traditional mining context to the notions used in the artifact-centric context. An artifact (type) is equivalent to the notion of a process definition, in which the event type definitions are specified. An artifact instance is instantiated from an artifact type. The artifact instances that share the same artifact type have similar event types. When extracting an event log based on a definition of an artifact type, each
artifact instance is similar to a process instance (or a case) which results in a trace of events.

To be able to reason about logs and to precisely specify the requirements for event logs, we use the formal definition of event log introduced by W. van der Aalst in the Process Mining book [2].

Let $E$ be the event universe, i.e., the set of all possible event identifiers. Events may be characterized by various attributes, e.g., an event may have a timestamp, correspond to an activity, is executed by a particular person, has associated costs, etc. Let $AN$ be a set of attribute names. For any event $e \in E$ and name $n \in AN$ : $\#_n(e)$ is the value of attribute $n$ for event $e$. If event $e$ does not have an attribute named $n$, then $\#_n(e) = \perp$ (null value).

The following standard attributes are used in this thesis: $\#_{\text{eventType}}(e)$ is the event type associated to event $e$. $\#_{\text{time}}(e)$ is the timestamp of event $e$ (also denoted by $T(e)$). $\#_{\text{resource}}(e)$ is the resource associated to event $e$.

Let $L$ be the case universe, i.e., the set of all possible case identifiers. A case also has attributes. For any case $c \in L$ and name $n \in AN$ : $\#_n(c)$ is the value of attribute $n$ for case $c$. If case $c$ does not have an attribute named $n$, then $\#_n(c) = \perp$ (null value).

Each case has a special mandatory attribute trace: $\#_{\text{trace}}(c) \in E^*$. We assume traces in a log contain at least one event, i.e. $\#_{\text{trace}}(c) \neq \emptyset$.

A trace is a finite sequence of events $\sigma \in E^*$ such that each event appears only once, i.e., for $1 \leq i < j \leq |\sigma| : \sigma(i) \neq \sigma(j)$. Thus, we also use $\sigma_c$ to denote $\#_{\text{trace}}(c)$.

An event log is a set of cases $L \subseteq L$ such that each event appears at most once in the entire log. We use $A_L$ to denote the set of all event types appearing in log $L$, i.e. $A_L = \{ \#_{\text{eventType}}(e) \mid c \in L \land e \in \#_{\text{trace}}(c) \}$.

A format for event logs is the XML based format XES\(^1\), which is selected as the standard format

---

\(^1\)http://www.xes-standard.org/
CHAPTER 2. PRELIMINARIES

for event logs by the IEEE Task Force on Process Mining. The complete meta model of the XES format is shown in Figure 2.3. The elements Trace, Event, and Attribute are comparable to the same element in the general event log structure introduced by Figure 2.2. For further detailed explanation, we refer to the official website of the standard. Both J. Buijs [9] and H. Verbeek et al. [28] have discussed the difference between the MXML event log format and the XES event log format. Since the XES format is an improved version based on the MXML event log format, well supported and chosen as the standard format, we use the XES event log in this thesis.

Figure 2.1: The UML 2.0 class diagram for the complete meta-model for the XES standard

Figure 2.3: XES meta model

Drawn using the UML 2.0 class diagram

2.1.2 Process Discovery and ProM framework

A specific type of techniques in the process mining domain is Process Discovery, which use an event log as input and discover a process model as output. Some well known process discovery techniques are Alpha algorithm [5], (Flexible) Heuristic miner [29][30], Genetic process mining [18], ILP mining [31], and Fuzzy mining [14]. Note that the model discovered from a given event log varies depending on which mining algorithm is used. This phenomenon is due to the fact that each discovery algorithm has its own definition to determine the causality between the event types. Another characteristic of discovery algorithms is that the algorithms are mainly aimed to identify the main flow and eliminate the outliers, whereas business users tend to use the discovered model to identify outliers. These aspects have to be taken into account for this thesis.

ProM is a generic open-source framework for implementing process mining tools in a standard environment [27][28]. The framework provides researchers an extensive base to implement new algorithms in the form of plug-ins. The framework provides an easy-to-use user interface functionality, a variety of model type implementations (e.g. Petri nets, EPCs) and common functionality like
reading and writing files. Each plug-in can be applied to any (compatible) object in the common object pool allowing the chaining of plug-ins to come to the desired result. ProM can read event logs stored in the new event log format XES (as explained in Section 2.1.1). Furthermore, it can also load process model definitions in a wide variety of formats. In this thesis, ProM 6 is chosen to implement the approach for the mining problem (II).

2.1.3 Process Models and Artifact-Centric Model

The process models discovered by process mining techniques describe the behaviors and the executions of real processes at an abstract level. Various process model notations are used to visualize these process models in various aspects such as the control flow aspect, or the organizational aspect. To describe the control flow aspect of processes, one can visualize the order of tasks to be executed by using model notations such as Transition Systems [6], Business Process Model and Notation (BPMN) [32], Petri nets [19], Event Driven Process Chains (EPC) [1], and Declare models [23]. To describe the organizational aspect of processes, one can visualize the hand-overs of tasks based on the categorization of resources using simple causal nets.

Within this thesis, we use the proclet system to describe the control flow aspects of a set of collaborative processes (denoted as artifacts) [4]. A proclet system consists of proclets and interactions found between the proclets. Each proclet describes the life cycle (i.e. control flow) of an artifact as a Petri net and its possible interactions with other artifacts, denoted by ports and channels. Each channel describes a uni-directional message flow from one proclet to another. A channel connects two proclets through ports of the two proclets. Each port of a proclet has a cardinality (C) indicating the number of other artifacts to which a message is sent or received from via the port. A value of a cardinality could be 1 (exactly once), ? (zero or once), * (zero or more) and + (once or more). An output port sends messages, and an input port receives messages which can be processed by a transition.

An example proclet system of the OTC example is shown in Figure 1.6. From left to right, each large rectangle represents a proclet showing the life-cycles of the artifacts Sales Order, Return Order, Delivery, Return Delivery and Invoice, respectively. The event type Created of the artifact Sales Order has an output port, which is connected via a channel with the input port of the event type Created of the artifact Delivery, whereas the event type Last Change of the artifact Sales Order has both an input port and an output port.

An event type level interaction between two artifacts is described by two ports and a channel connecting the two proclets. The event level interactions can be compared to the messages which are sent via the channel.

2.1.4 Relational Databases and ERP systems

The data sources in the research scope of this thesis are the relational data stored in relational databases by information systems. Relational databases are a shared collection of logically related data (tables) described by a relational model (e.g. the OTC example illustrated in Figure 1.1). In other words, software systems which store the transactional data related to business executions in a relational database can be used as input. For example, Enterprise Resource Planning (ERP) systems, known as software systems that support the optimal usage of all resources in an organization, or the
work flow management systems (WfMS), which already have clear definitions of processes supported by the system, or Customer Relational Management (CRM) systems, which manage the data related to customers. Moreover, we assumed that the input data source contains information about the documents in a process, their relations to each other and timestamps of the tasks performed on these documents that are relevant for the process analysis.

Within this thesis, the data sources used to evaluate the new proposed approach are relational data created and used by ERP systems: the Order To Cash (OTC) process supported by SAP, and the Purchase to Pay (PTP) and Project Administration (PA) processes supported by the Oracle ERP system.

2.2 Literature Study - Traditional Log Extraction Approaches

The introduction to event logs and relational data has shown that both data structures differ from each other fundamentally. One of the main differences is that a process definition has a clear notion of a case, whereas the relational data contains several notions of “cases” (i.e. different objects or documents). Therefore, before process mining techniques can be applied, the relational data has to be converted to an event log or multiple event logs. In this section, we first explain the characteristics of traditional event log extraction approaches. Then, several traditional approaches found in the literature are discussed briefly. Finally, we will summarize the issues related to the traditional approaches in general which led to the new artifact-oriented approach.

Traditional log extraction approaches are approaches which extract event logs from data-centric systems based only on one notion of a case. These approaches first (try to) identify or define one notion of a case. After specifying or selecting the event types (manually), the approaches collect the events found in the data source that are associated with the cases found for the defined case notion, e.g. the way we created the event log shown in Figure 1.3. Having all relevant data collected, the approaches rewrite the data as one event log in a log format such as XES. These approaches only extract one log for one process definition at a time, while assuming the process is isolated and has no interaction with other processes or its system environment.

M. van Giessel is the first who investigated the applicability of process mining in ERP systems [13]. He proposed a manual approach for applying process mining on SAP which consists of three steps and uses the SAP reference model as a guide: (1) find relevant tables containing the data for the event log; (2) find the relationships between the relevant tables; (3) find event types related to the document identifiers found in the relevant tables. The event log is then created by hand which is very laborious for a large number of events.

I.E.A. Segers used ProM 5 and the ProM import framework to create an event log from SAP R/3 system for auditing purposes, which still requires a great amount of manual work [26]. The event log was created in an MXML log format. According to D. Piessens, Segers was one of the first who pointed out the convergence and divergence problems. The convergence issue is defined as the situation when an event is linked to multiple process instances (or cases). The divergence issue is defined as the situation when multiple events of the same event type are linked to one case. These two known issues are caused by the lack of a clear notion of a case.

J. Buijs was the first one who proposed a general approach to extract event logs in the XES format from various data sources [9]. Moreover, Buijs has built a tool, named XESame, to support the
conversion. By first creating a conversion definition specifying which part of the source data should be mapped to which elements in the resulting event log (e.g. the traces, events and attributes), the conversion definition is then used to create SQL queries which are again used to build the cache database to store the data in a certain structure. The cache database is then used to construct event logs. However, creating a conversion definition is time consuming and requires domain knowledge, SQL querying and process mining knowledge, since the starting point of the conversion definition is the log structure. In Buijs’ thesis, the concept of convergence and divergence are again described. He proposed an approach to partially solve the data convergence and divergence problems: choose a lower level case identifier. However, this solution can not be applied to every situation, e.g. the aforementioned OTC example does not have a lower level.

D. Piessens developed a new tool, named SAP Log Extractor, specifically for the Order to Cash (OTC) and the Purchase to Pay (PTP) processes supported by SAP [24]. Instead of supporting users with creating conversion definitions, Piessens predefined a repository which contains event types and attributes definitions of the processes. Users can select a set of event types which are desired from the set of predefined event types. The selected set of event types is used to determine a list of valid case identifiers (i.e. primary keys) and different ways to link the tables containing the event types to the case identifier based on the foreign keys. A possible case identifier and the corresponding links are denoted as a case-table mapping. A list of case-table mappings is provided to users to allow them select the desired one. However, the process repository is created manually, which is also very time consuming. With respect to convergence and divergence problems, Piessens first proposed the same approach as Buijs to select a different case identifiers (e.g. instead of a document header, choose a document line to be a case instance). However, this does not solve the convergence and divergence. As second solution, Piessens briefly introduced the possibility of artifact-centric process models as further research.

A. Roest proposed the idea of combining several (event) identifiers as unique case identifier to solve the data divergence problem [25]. He manually built an SQL query for the OTC process of SAP to extract logs comprising traces that have a case identifier composed of the primary keys of all relevant tables. Unfortunately, this solution obstructs the statistics and was unable to solve the data convergence problem. Moreover, the notion of case varies within a single process definition.

Furthermore, both Buijs [9] and Piessens [24] discussed several commercial tools such as Enterprise Visualization Suite (EVS) distributed by Businessscape, the ARIS Process Performance Manager by ARIS, LiveModel by Intellicorp, and Nitro developed by Fluxicon. We have also found a commercial tool for SAP, SNP Business Process Extractor developed by SNP-AG. We assume that these tools use the traditional approaches, i.e. are unable to deal with data divergence and convergence problem and unable to detect interactions between different processes. This assumption is made based on our experiences with Nitro, the discussion found in the literature about the tools and a presentation of SNP-AG.

We have already mentioned several issues of the traditional log extraction approaches. First, the data divergence and convergence are addressed but not solved completely. The many-to-many relation between the sales orders and invoices illustrated by the OTC example has shown that the solutions (e.g. choose a different case notion such as using the items in the documents instead of the documents, or choose a different foreign key to relate events) proposed by the traditional log extraction approaches can not solve both divergence and convergence. Second, the traditional log extraction approaches are very scenario and situation specific. Most traditional log extraction approaches are specifically designed for the standard processes (e.g. OTC and PTP) of SAP, which
means extending these approaches to other systems or processes will require a great amount of work. J.Buijs [9] is the only one who has implemented a general solution to support log extractions from data sources. However, users still have to manually define the conversion definition (i.e. a mapping definition between the data source and the event log). Based on the experience of the OTC process of SAP, it is observed that the ‘hard’ foreign keys between the tables of large ERP system may remain the same, but the ‘soft’ relations vary between different system instances. This variety means the relation between the event types and the case might change for each instance of SAP, which means that the conversion definition has to be redefined. For instance, we consider the sales orders as cases, and the creation of return deliveries as events. Now, without changing the foreign keys, we can propose a new example, in which the return delivery has a foreign key to sales orders instead of return orders, thus changing the reference SD id of the return delivery D4 from S3 to S1. Now, the mapping definition of the return delivery events have to be changed. These issues have led to the development of another type of approach: the artifact-centric approach.

2.3 Literature Study - Artifact-Centric Log Extraction Approach

We have discussed that the traditional log extraction approaches have encountered many problems such as data convergence and divergence when applied within ERP systems, since many information systems use relational databases which are data-centric and document-centric causing ambiguous case definition. Therefore, a different type of log extraction approach, named artifact-centric approach, is proposed by E. Nooijen et al. [21][22] to model a process as the life cycles of several objects and interactions between the objects. In this section, we first recall the motivation to apply artifact-centric approach. Then, we give a brief overview of the original XTract approach. In Section 2.3.1, we discuss each step of the original XTract approach in more detail with the issues identified when applying the XTract approach on real ERP systems, which are also the reasons for extending the XTract approach.

We have argued that the artifact-centric approach is a more suitable way for extracting event logs within data-centric systems than the traditional log extraction approach. The new conceptual view provided by the artifact-centric approach allows us to consider a process as several collaborative artifacts with their own life-cycles, instead of fixed to the assumption (also made by traditional log extraction approaches) that a process is only related to a single notion of a case, and it is isolated and has no interaction with its environment. This view is more suitable for data-centric systems, since in data-centric systems, we have also various interrelated objects, e.g. sales orders, deliveries, invoices, and each has its own life-cycles. Furthermore, the notion of interactions between artifacts introduced by the artifact-centric approach allows us to express one-to-many and many-to-one causalities between artifacts that caused the data divergence and convergence problems in the traditional log extraction approach. Solving the data divergence and convergence problems help us to obtain a simpler and more intuitive model and thus increase the possibility to discover true-positive unusual flows. Finally, the artifact-centric approach actually generalizes the traditional log extraction approach because we can also consider a large process also as one artifact.

The concept of artifacts is first introduced by A. Nigam and N. Caswell [20] and put into practice by D. Cohn and R. Hull [10]. However, artifacts were created manually and based on interviews, which is time-consuming. E. Nooijen et al. [21][22] were the first who proposed an artifact centric log extraction approach, named XTract, to fully automatically extract the event logs from a given
The overall method of the original XTract data source. The XTract approach was tested using the developed prototype and conducting case studies.

The overall method of the XTract approach consists of two phases, Artifact Schema Identification and Artifact Lifecycle Discovery, which are divided in five steps, as shown in Figure 2.4. First, (1) given a relational data source, the schema of the source is automatically extracted. Second, Nooijen (2) uses the schema extracted to identify artifact schema(s). Third, (3) a schema log mapping is automatically created for an artifact schema representing one single isolated artifact which is selected by users. Fourth, (4) the schema-log mapping (which is similar to a conversion definition used by J. Buijs) is then used to create SQL queries. The queries are used to build a cache database to temporarily store the data of event log, which is then used to automatically generate event logs. As the last step, (5) the XTract reused existing process discovery techniques to mine the life cycles of an artifact.

Since the XTract approach is the only general approach found in the literature and the source code is made available. We used the XTract approach as the start point of our research. We investigated the XTract approach in detail by applying the XTract tool on the available data.

### 2.3.1 Open Issues of XTract

By applying the XTract approach to the OTC example of Figure 1.1, we can highlight the open issues in XTract that make it currently inapplicable in practice. In this section, we discuss these open issues in detail regarding each step of the XTract approach, introduce our solution to the issue and refer to the specific section in which our approach is discussed more in detail.
CHAPTER 2. PRELIMINARIES

Given a relational source, the XTract approach assumes that no knowledge about this relational source is available. Therefore, the first step is to (1) automatically identify the data schema consisting of primary keys, foreign keys, column domains and a function assigning a column to a domain. Nooijen used existing schema identification techniques such as MIND [16][17] and SPIDER [8][7] for the foreign key identification. However, as the literature shows, these methods can only identify hard foreign key dependencies, i.e. each tuple value in the child columns should match to a value tuple in the parent columns such as the foreign key between (the SD id column of) the SD table and (the Reference SD id column of) the DD table (i.e. $F_2$ in Figure 1.1) of the OTC example.

Unable to Identify Complex Foreign Keys

The XTract approach was unable to automatically identify all foreign keys. Applying the first step (1) of the XTract approach on the OTC example, the result in Figure 2.5 shows that the XTract approach is only able to identify the foreign key between (the SD id column of) the SD table and (the Reference SD id column of) the DD table (i.e. $F_2$ in Figure 1.1). The other three foreign keys are omitted. E. Nooijen has also indicated that the schema identification is an NP-complete problem, and available solutions take exponential time in the number of columns for identifying the schema. The case studies of Nooijen have also shown that there are severe run-time issues, e.g. unable to terminate when identifying 2 column foreign keys for large ERP systems. Since the XTract tool did not provide any support for adding foreign keys or importing foreign keys, we had to manually add the other foreign keys in the database to perform the subsequent steps. During the collaboration with KPMG advisers, it had been noticed that the advisers already have knowledge of the schema of the databases including foreign keys, or the foreign keys might be documented for large ERP systems. Therefore, it might be faster and easier to support users by importing and reusing the schemas. This solution is used in our approach and further explained in Section 3.1.

We have also noticed that the data structure used by the XTract approach for foreign keys does not allow to express the foreign key between the Document Changes and other three tables (i.e. $F_4$ in Figure 1.1) of the OTC example, where the column Table name should have a certain value. We also solved this limitation by extending the definition of reference which is explained in Section 3.1.

Convergence and Divergence Problems

Having the schema of the data source identified, the XTract approach performed the second step (2) artifact schema(s) identification as follows. Based on the importance of the table (i.e. entropy [11])

![Figure 2.5: The foreign key of OTC example that is identified automatically](image-url)
and the distance of the references (i.e. the matched average fraction [33]), k-clustering is applied to
cluster tables into different artifact schemas. The number \( k \) denotes the number of artifact schema
and is specified by users. The ‘center’ (according to the defined distance) is automatically returned
as the main table of the artifact schema containing the case instances. As the calculation of distance
of the references have not taken into account that relations other than one-to-one (i.e. the matched
average fraction is greater than 1) should be infinite, two tables linked by relations other than one-
to-one are not explicitly clustered into two artifacts schemas. Using the OTC example, we obtain
the following artifact schemas.

- For \( k = 1 \), the XTract approach returned an artifact schema consisting all four tables with the
  main table as the Document Changes table.
- For \( k = 2 \), the XTract approach returned two artifact schemas, one consists of the BD table
  (as main table), and the other consists of the SD, DD and the changes table with the main
table as the Document Changes table.
- For \( k = 3 \), three artifact schemas were returned. One consists of the BD table (as main table);
  One consists of the Changes table (as main table); and the other consists of the SD and DD
tables and the main table is unknown.
- For \( k = 4 \), four artifact schemas were returned. One consists of the BD table (as main table);
  One consists of the Changes table (as main table); one consists of the SD and DD tables and
  the main table is unknown; and the fourth is an empty artifact schema.

The results show that the k-clustering approach leaves the convergence and divergence within an
artifact schema unaddressed. Moreover, the results also show that it is very unintuitive to cluster the
tables of OTC to the artifact schemas this way. It is more logical to obtain three artifact schemas
for the OTC example: one with table SD in (and table Changes), one with table DD (and table
Changes), and one with table BD (and table Changes) to indicate the sales documents artifacts,
delivery documents artifacts and billing documents artifacts, respectively. Therefore, to be able to
perform the next step, the three artifact schema are manually inserted in the database. In our
approach, we present a new algorithm to identify the artifact schemas, which does solve the data
convergence and data divergence problem. Moreover, we allow users to change the artifacts schemas
identified, which is explained in Section 3.2.

Unable to Identify Artifacts within One Table and Event Types within One Column

During the third step (3) schema-log mapping creation, an artifact schema that is identified during
the previous step is automatically mapped to one artifact, based on which a log mapping (i.e.
one conversion definition) is created. The primary key of the main table of an artifact schema is
mapped to the artifact instance identifier and is mapped to the traceID of the TraceMapping; the
main table of the artifact (schema) is the fromTable of the TraceMapping; each time column in
the artifact schema is automatically mapped to an event type of the artifact, which is mapped to
an EventMapping in the mapping; an attribute is either mapped to an AttributeMapping if it has
only one value or mapped to an ListAttribute if it has multiple values. Note each column used has
already defined the XesAttribute key, type and extension during the domain calculation. For the
artifact Sales Documents, we obtain the following mapping shown in Figure 2.6.

For the third step (3) schema-log mapping creation, we have noticed the following limitations.
First, as the OTC example shows, the sales document Sales Order and the sales document Return
Order have many differences. For example, return order (S3) has a reference to sales order S1 but sales orders does not have any references to return order; return order (S3) does not have last change or any changes in the Document Changes table. Similar differences could be found for the Deliveries and Return Deliveries. Therefore, it might be interesting and required by users to be able to consider the sales orders and return orders as different artifacts. However, the XTract approach does not support this since the XTract approach assumes the instances in a main table all belong to one artifact. Second, as Figure 2.6 shows, the Date changed column is automatically considered to be one event type by the XTract approach. However, the Change type column indicates different types of changes which have different values and properties. Thus, users might be interested in analyzing different changes as different event types. However, this is also not supported by the XTract approach. Third, the artifact and the mapping are automatically created by the XTract approach while selecting all event columns and other attribute columns. Users are unable to select the event types or attributes that are relevant. Selecting all non-event columns as attributes has also led to performance issue when writing the log.

Our approach, described in Section 3.3, solved this issue by extending the definition of an artifact and an event type. Then, we proposed a semi-automatic approach to identify multiple artifacts sharing the same main tables, and also identifies the event types share the same time column. Furthermore, we added the functionality to allow users change the artifact identified.

Performance Issues Related to Log Extraction

Having the log mapping created, the original XTract approach (4) generates traces by deploying the same basic idea as XESame: the log mapping (i.e. a conversion definition) is used to automatically write SQL queries, which are again used to automatically build the cache database to store the data in a certain structure. The cache database is then used for wring an XES event log file. The data structure of the cache database is simple: the trace ids are stored in the XTrace table; the event ids are stored in the XEvent table; and all attributes of the log, traces, events and attributes are stored in the XAttributes table. Finally, OpenXes is used by the original XTract approach to write an XES event log file. We found performance issues when writing logs from the cache database. We suspect the performance issues is an implementation issue which is related to the data structure chosen for the cache database. Unfortunately, we were unable to further investigate or to solve the performance issues due to the time constraint.
No Interaction Identified and Other Issues

Another issue found is that the method used by the XTract approach for creating the SQL queries was unable to deal with foreign keys with self loops, e.g. the foreign key from the SD table to the SD table (i.e. $F_1$ in Figure 1.1) in the OTC example. Assume in the OTC example, we have changes in the Document changes table that refer to previous changes of a sales document but not direct linked to the sales document, these change events will not be able to be identified.

No interactions are identified between the different artifacts by the XTract approach. The missing interactions are one of the most interesting aspects of applying artifact-centric approach since otherwise, the artifact-centric approach only returns a set of single isolated trivial processes. For example, no information about the relations between the sales orders and deliveries can be obtained. Our approach addresses the identification of interactions between artifacts in Chapter 4.

Nooijen uses the existing process discovery techniques to discover the life-cycles of artifacts. However, no interaction between (the event types of) the life-cycles is discovered by XTract. We proposed two solutions to identify interactions between event types and thus discovering the artifact-centric model, discussed in Chapter 5.
Chapter 3

Artifact Type Identification

In Chapter 1, we have defined our research problem and research scope as obtaining an artifact centric model from a data source. We have divided the problem into three phases, (I) extraction, (II) mining and (III) evaluation. In this chapter, we shall introduce our solution to the first step of the extraction phase: (A) artifact type identification. In other words, given a data source, we would like to be able to identify a set of artifacts which divide the data source into collections of data, each of which contains complete information of an artifact. Each of the returned artifacts has a set of trace id columns, event type columns and attribute columns, which is aligned with the structure of an event log. For now, we do not consider any interactions between artifacts. We first recall the reason why the original XTract approach is not applicable in practice, followed by a short overview of our approach, in which we will also introduce the rest sections of this chapter.

The original XTract approach is not suitable for creating the desired artifacts in practice. First, the complex foreign keys can not be expressed with the current data structure and can not be identified automatically (or identified within an acceptable time frame). Moreover, the way in which XTract creates artifacts does not explicitly exclude one-to-many and many-to-one relations within an artifact, and thus leaving the convergence and divergences problems unsolved. Furthermore, we found that instances within a table can belong to separate artifacts, whereas the XTract approach assumes that a main table only contains instances of one artifact. Also, events of different event types could share the same event column, which is not supported by XTract. Finally, the XTract prototype does not support manually changing artifacts which automatically includes all event type and attribute columns.

To create desired artifacts for real-life ERP systems, we proposed a new approach extended based on the XTract approach. Our approach follows three steps using the same numbering as in Section 1.4: (A1) data schema identification, (A2) artifact schema identification, and (A3) artifact identification, of which an overview is shown in Figure 3.1. In the following sections, we shall emphasize the functions added and using examples to illustrate the difference between the XTract approach and our approach. First, given a data source, we now also allow to selectively import a part of the tables of the data source (1.1). While in the original XTract one can not import the primary keys and foreign keys separately, we now also allow to import this partial schema information (1.2) to be able to identify complex data schemas, described in Section 3.1. Second, in Section 3.2, we present our algorithm which identifies artifact schemas by fully splitting the tables based on references to explicitly avoid the convergence and divergence problem (2.1). In contrast to the original XTract
CHAPTER 3. ARTIFACT TYPE IDENTIFICATION

1. Artifacts Identification

Import
- Fully auto detection (1,1)
- Selective Import (1.2)
- Import K-clustering (2.1)
- Full Split (2.2)

Support changes
- One artifact creation (3.1)
- Multiple artifacts creation (3.2)

Primary keys
- Foreign keys

Data sources
- Tables
- Data schema
- Artifact schemas
- Artifacts with conditions

Figure 3.1: Method Artifact Identification

The black arcs show the existing functions of the XTract approach, whereas the red arcs show the methods added.

approach, we also support users to add tables to or remove tables from a specific artifact schema to adjust which data is considered during the artifact identification (2.2). In Section 3.3, having identified the artifact schemas, we extended the definition of an artifact to an artifact with condition to allow identifying multiple artifacts sharing the same main table (also known as instance table) (3.1), instead of mapping an artifact schema to only one artifact, which is done by the original XTract. In addition, our method provides users support to add, change and remove each artifact identified (3.2).

3.1 Data Schema Identification

In this section, we introduce the step (A1), data schema identification, of the artifact identification to discover the data schema including primary keys, foreign keys, and column domains from a given data source. We first illustrate the issues of the reference definition used by the XTract approach. Then, we generalized the definition of references to express complex foreign keys, followed a formal definition of the data schema identification problem. Finally, we present our solution which identifies data schemas by allowing users to import primary keys and reference. The data schema can be used to later identify event types, case identifiers and other elements to create event logs.

The fully automatic (data) schema identification used by the XTract approach has encountered several issues such as being unable to identify complex reference keys and long computation time. During the collaboration with KPMG advisers, it had been noticed that the advisers already have knowledge of the schema of the databases including reference keys, or the reference keys might be documented for large ERP systems. Therefore, the method should allow users to express the schema and be able to export and import the schema to reuse it for ERP systems which share the same structure and tables. However, we notice the reference definition $F_{original} = (T_p, C_p, T_c, C_c)$ used by the original XTract is unable to express complex references. For instance, if the $SD$, $DD$ and $BD$ tables of Chapter 1 overlap in primary key values such as when the first letter is removed from the identifiers shown in Figure 3.2, then the condition that the Table name in the CHANGES table
should be ‘SD’ is a crucial part of the reference when joining the SD tables. Without this condition, the CHANGES table would also relate changes to a sales order that actually relate to an invoice and vice-versa. Another complex reference structure observed in the pricing procedure tables (e.g. KONV) of SAP is that a column could obtain a combined value that are linked to multiple tables. The change record 5 shown in Figure 3.2 illustrates such a complex reference.

We redefine a reference $F = \langle T_p, C_p, T_c, C_c, F_{condition} \rangle$ as a 5-tuple consisting of a parent table $T_p$, an ordered subset $C_p$ of columns denoting the primary key of the parent table, a child table $T_c$, an ordered subset $C_c$ of columns denoting the foreign key, and an extra condition $F_{condition}$ for the reference (which can be appended in the FROM part or the WHERE part of an SQL query). For instance, the condition $F_{condition}$ could be empty indicating $F_{condition}$ is true, or it could indicate that a certain column of the parent table should have a certain value. The references of the OTC example, some of which are shown in Table 3.1, are given in the corresponding definition format. For example, the reference $F_4$ is split in different references with the corresponding condition $F_{condition}$.

Figure 3.2: The counterexample OTC for references

The problem of data schema identification can be formally defined as follows. Let $T = \{ T_1, \ldots, T_n \}$ be a set of tables imported from the data sources, where each table $T_i = \langle C, C_p \rangle$ is a tuple of its columns $C$ and its primary keys $C_p = \emptyset$. Assume that the knowledge of primary keys and foreign keys is available, which can either be detected using the original XTract approach or can be imported by the new method as a set $P$ of primary keys and a set $F$ of references, where each $P = (T, C_p) \in P$. We would like to compute a schema $S = (T,F,D,\text{column\_domain})$, which is defined as a 4-tuple of a set $T$ of the tables with the primary keys $C_p$ of each table $T_i = \langle C, C_p \rangle \in T$ filled, a set $F$ of references, a set $D$ of domains, and an assignment function $\text{column\_domain}$ which returns the domain of the given column.

Our method solves schema identification as follows. First, the domain extraction functions provided by the XTract approach are reused to obtain the domains $D$ and the domain assignment function $\text{column\_domain}$. Second, since the new method allows users to import the primary keys
CHAPTER 3. ARTIFACT TYPE IDENTIFICATION

Table 3.1: Reference keys

<table>
<thead>
<tr>
<th>Parent table ( T_p )</th>
<th>Parent columns ( C_p )</th>
<th>Child table ( T_c )</th>
<th>Child columns ( C_c )</th>
<th>Condition ( F_{condition} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>SD id</td>
<td>SD</td>
<td>Reference id</td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>SD id</td>
<td>DD</td>
<td>Reference SD id</td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>SD id</td>
<td>CHANGES</td>
<td>Reference id</td>
<td>( \text{CHANGES, [Table name]} = \text{’SD’} )</td>
</tr>
<tr>
<td>BD</td>
<td>BD id</td>
<td>CHANGES</td>
<td>Reference id</td>
<td>( \text{CHANGES, [Table name]} = \text{’BD’} )</td>
</tr>
<tr>
<td>SD</td>
<td>∅</td>
<td>CHANGES</td>
<td>∅</td>
<td>( \text{left(CHANGES, [Reference id], 1)} = \text{SD.[SD id]} \text{ AND left(CHANGES, [Table name], 2)} = \text{’SD’} )</td>
</tr>
</tbody>
</table>

\( C_p \) for each table \( T_i \in \mathbb{T} \) and the reference keys \( F \) which satisfy the new reference definition, we have obtained all components which are sufficient to return the schema. Note it is also possible to automatically detect primary keys and foreign keys by using the original XTract approach and use these keys as input, since the definition of primary key is unchanged and the foreign key definition used by the XTract approach can be easily converted as the new reference by setting the condition to true.

3.2 Artifact Schema Identification

In this section, we present the second step, (A2) artifact schema identification, in which we proposed a new algorithm to identify artifact schemas based on the given data schema and prevent the convergence and divergence problem. Since the artifact schemas later determine the structure of artifacts, this steps should solve or prevent the convergence and divergence problem in general.

The k-clustering approach, used by the original XTract approach to calculate table clusters where each cluster represents an artifact schema, does not solve the data convergence and data divergence problem identified throughout the literature study, since the one-to-many relations and many-to-one relations which cause these problems are not explicitly excluded within an artifact schema. When an artifact of such an artifact schema is mapped to an event log, a one-to-many relation between the main table containing the case instances and an event type column will lead to the same event type being executed on one process instance (i.e. a case instance) multiple times. And a many-to-one relation between the main table containing the case instances and an event type column will lead to the same event being executed on multiple process instances at once. Because the artifact schema of an artifact determines the data used to construct the artifact, the one-to-many and many-to-one relations between the table of artifact instance and the event type columns should be explicitly excluded from an artifact schema.

Therefore, we present a new artifact schema identification method that uses the schema \( S = (\mathbb{T}, \mathbb{F}, \mathbb{D}, \text{column\_domain}) \) as input and returns a set \( \mathcal{S} = \{ \mathcal{S}_{A_1}, \ldots, \mathcal{S}_{A_n} \} \) of artifact schemas, where an artifact schema \( \mathcal{S}_{A} = (\mathbb{T}_A, \mathbb{F}_A, \mathbb{D}_A, \text{column\_domain}, T_m) \) is a subset of the schema \( \mathcal{S} \), defined as a 5-tuple which comprises a subset \( \mathbb{T}_A \subseteq \mathbb{T} \) of tables, a subset \( \mathbb{F}_A \subseteq \mathbb{F} \) of references, a subset \( \mathbb{D}_A \subseteq \mathbb{D} \) of domains, the assignment function \( \text{column\_domain} \) of the schema, and a table \( T_m \in \mathbb{T}_A \) denoting the instance table in which the trace identifiers can be found. An artifact schema
comprises complete information related to its artifact(s). Note the definition of an artifact schema is unchanged from the original XTract approach, which means the original XTract approach can still be used to detect artifact schemas.

The new simple algorithm \textit{ComputeArtifactSchemas}(\mathcal{S}) shown below identifies artifact schemas while preventing the data divergence and convergence problem. Since the one to many and many to one references cause the divergence and convergence problem, these references are not desired within an artifact schema. Therefore, if we construct a schema graph where each table is a node, and each reference is an edge from the parent table to the child table, then we can remove the references which are not one to one, thus resulting in a graph only connected by one to one reference. The sub graphs which are still connected can be considered as valid artifact schemas as it only contains tables which are linked by one to one references. The main table \(T_m\) can be selected as a table which has no parent in the set \(T_A\) of the selected tables. The set \(\mathbb{D}_A\) of domains is the union set of all domains of columns of the tables in \(T_A\). We can obtain an artifact schema \(\mathcal{S}_A = \langle T_A, F_A, \mathbb{D}_A, \mathcal{S}.column\_domain, T_m \rangle\) and add it to the set \(\mathcal{S}\) to be returned.

\begin{algorithm}[h]
\caption{ComputeArtifactSchemas(\mathcal{S})}
\begin{algorithmic}
\STATE Let a graph \(G = (T_G, F_G) \leftarrow (\mathcal{S}.T, \mathcal{S}.F)\)
\FOR {\(F \in F_G\)}\IF {\(F\) is not one to one}\STATE remove \(F\) from \(F_G\)\ENDIF\FOR {a connected sub graph \(g = (T_g, F_g) \subseteq G\)}\STATE select a table \(T_m \in T_A\) which has no parent table in \(T_A\)\STATE \(\mathbb{D}_A \leftarrow \) the union set of domains of columns of the tables in \(T_A\)\STATE create a new artifact schema \(\mathcal{S}_A = \langle T_A, F_A, \mathbb{D}_A, \mathcal{S}.column\_domain, T_m \rangle\)\STATE add artifact schema \(\mathcal{S}_A\) to \(\mathcal{S}\)\ENDIF\ENDFOR\ENDFOR\STATE \textbf{return} \(\mathcal{S}\)
\end{algorithmic}
\end{algorithm}

The algorithm \textit{ComputeArtifactSchemas} presented, which is a simple brute-force way of splitting tables to exclude one-to-many relations, also faces some issues. For example, in SAP, the change tables \textit{CDHDR} and \textit{CDPOS} contain all change records of the sales documents stored in the table \textit{VBAK}. It might be more logical to consider the changes as a part of the life-cycle of the \textit{Sales Order} artifact, instead of as a separate artifact. However, the change tables are returned as a separate artifact schema by the algorithm because the sales document table is linked to the change tables by a one-to-many reference, which means a sales document could be linked to multiple change records (i.e. data divergence). However, a ‘perfect’ artifact (schema) has not been defined. The tables assigned to an artifact schema influence the data contained in the event log. Therefore, our method allows users to add tables to or delete tables from the artifact schemas to construct an entity of tables containing all information related to an artifact which shall be subsequently converted to an event log that satisfy the requirements of users. Thus, users can decide whether to consider the changes as a separate artifact (then there are interactions between sales order and changes which solves the divergence problem) or as a part of the \textit{Sales Order} artifact’s life-cycle by adding the \textit{Changes} table to the artifact schema of SD (which cause the divergence problem). Furthermore, users can construct artifact schemas which overlap in tables.

We shall illustrate the different between the XTract approach and our artifact schema identification. Given the OTC example, the XTract approach returns the three artifact schemas shown
in the left table of Figure 3.3 when given $k = 3$ (as discussed in Section 2.3). Our approach shall first return three artifact schemas SD, BD, and DD as shown by the black tables in the right table of Figure 3.3. Since only one invoice has a change, the document changes table is assigned to the BD artifact schema. The SD artifact schema returned only contains the SD table, similar for the DD artifact schema. Now if users desire to include changes for the SD artifact, they can add the changes table to the SD artifact schema. This comparison shows that it is more intuitive and easier to apply the new approach.

<table>
<thead>
<tr>
<th>XTract: Artifact Schemas (k = 3)</th>
<th>Our Approach: Artifact schemas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Main table</td>
</tr>
<tr>
<td>BD</td>
<td>BD</td>
</tr>
<tr>
<td>Changes</td>
<td>Changes</td>
</tr>
<tr>
<td>Changes</td>
<td>?</td>
</tr>
</tbody>
</table>

Figure 3.3: Comparing the artifact schemas obtained using the XTract approach and our approach with respect to tables $T$ and the main table $T_m$.

### 3.3 Artifact Identification

In this section, we present the third step, (A3) artifact identification, in which we have extended the definition of artifacts and event types to be able to identify multiple artifacts of which the case instances share the same main table and multiple event types of which the events share the same time column. Furthermore, we present a semi-automatic method to help users identify these artifacts and event types, when given a set of artifact schemas.

Our artifact identification method should be able to identify multiple artifacts sharing the same artifact schema, since the assumption made by the original XTract approach that a main table table $T_m$ of an artifact schema $S_A$ only contains the instance information of one specific artifact is unrealistic. We use the OTC example as a counterexample. In the $SD$ table shown in Figure 1.1, multiple document types that share the same data structure are stored: two sales orders and one return order. The return order has a reference to the first sales order with id S1, whereas the sales orders have no reference to previous documents. In addition, the return order S3 has a reference with the return delivery and no reference with the deliveries in the DD table, whereas the sales order do have references with the deliveries and no direct reference with the return delivery. This information already indicates the sales order and the return order have different interactions with the system. Therefore it is important to be able to distinguish the two document types as different artifacts having separate life cycles, such that the interactions could also be identified separately.

After the artifact schemas $S_A$ are identified, the artifact identification method, which uses each of these artifact schemas to obtain one or multiple artifacts, is defined as follows. Given an artifact schema $S_A = (T_A, F_A, D_A, column\_domain, T_m)$, the method returns a set of artifacts, where each artifact $A = (A_{name}, C_{Aid}, E, C_{attrs}, I, S_A, A_{condition})$ denotes a 7-tuple which comprises a name $A_{name}$, a set $C_{Aid}$ of columns denoting the case identifier of the artifact, a set $E$ of event types, a set $C_{attrs}$ of columns denoting the case attributes, a set $I$ of interactions between this artifact $A$ and other artifacts (which is an empty set for now), the corresponding artifact schema $S_A$, and an artifact condition $A_{condition}$ which is an extra condition (which can be appended in
### Table 3.2: An example of the Sales Order artifact

<table>
<thead>
<tr>
<th>Artifact’s component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{name}$</td>
<td>Sales Order</td>
</tr>
<tr>
<td>$C_{Aid}$</td>
<td>{SD id }</td>
</tr>
<tr>
<td>$E_1 \in E$</td>
<td>$(E_{name} = \text{date created},\ E_{Eid} = {SD id},\ E_{time} = \text{date created},\ E_{Eattrs} = {},\ E_{condition} = \emptyset)$</td>
</tr>
<tr>
<td>$E_2 \in E$</td>
<td>$(E_{name} = \text{last change},\ E_{Eid} = {SD id},\ E_{time} = \text{latest change},\ E_{Eattrs} = {},\ E_{condition} = \emptyset)$</td>
</tr>
<tr>
<td>$C_{attrs}$</td>
<td>{ [Document type], [Value] }</td>
</tr>
<tr>
<td>$I$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$A_{condition}$</td>
<td>$T_m, [\text{Document type}] = \text{’Sales Order’}$</td>
</tr>
</tbody>
</table>
CHAPTER 3. ARTIFACT TYPE IDENTIFICATION

... AND $C_n = v_m$. In addition, for the columns in $C$ which are not in the main table $S_A.T_m$, the tables of these columns should be selected to be joined with the main table such that the condition can be tested on these columns. In this case, the shortest path algorithm can be used to select references and the tables. As a result, a set of complete artifacts without interactions is constructed by $createArtifactsByColumnValues$.

It is also possible to import a predefined list of artifacts using the function $importArtifact(T'_m, A_{name}, A_{condition})$. For each artifact schema $S_A$ retrieved where the main table $T_m$ of $S_A$ is equal to $T'_m$, the function constructs an artifact by calling the function $createArtifact(S_A, A_{name}, A_{condition})$.

Figure 3.4 demonstrates the difference in artifacts returned by the XTract approach and our approach. Given the artifact schema SD containing the table SD as the main table and table Changes, the XTract approach returns one artifact SD shown on the left hand side in Figure 3.4. In contrast, the user can indicate the document type column as a condition column constituting $C_e$ in the function $createArtifactsByColumnValues(S_A, C_e)$. Two artifacts Sales Order and Return Order are then identified, as shown on the right hand side in Figure 3.4.

<table>
<thead>
<tr>
<th>Given Artifact schema SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>SD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The original Xtract approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>one $T_m$ $\rightarrow$ one artifact</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Artifact name</th>
<th>Artifact identifier</th>
<th>Event type</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SalesDocuments</td>
<td>SD id</td>
<td>DateCreated</td>
<td>$SD.[Document type] = 'Sales Order'$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Event id</td>
<td>$SD.id$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Timestamp</td>
<td>$date created$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>one $T_m$ $\rightarrow$ multiple artifacts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Artifact name</th>
<th>Artifact identifier</th>
<th>Event type</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales Order</td>
<td>SD id</td>
<td>DateCreated</td>
<td>$SD.[Document type] = 'Sales Order'$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Event id</td>
<td>$SD.id$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Timestamp</td>
<td>$date created$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Artifact name</th>
<th>Artifact identifier</th>
<th>Event type</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Order</td>
<td>SD id</td>
<td>DateCreated</td>
<td>$SD.[Document type] = 'Return Order'$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Event id</td>
<td>$SD.id$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Timestamp</td>
<td>$date created$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event type</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateCreated</td>
<td>$SD.[Document type] = 'Sales Order'$</td>
</tr>
<tr>
<td>PriceUpdated</td>
<td>$Price updated$</td>
</tr>
<tr>
<td>DeliveryBlockReleased</td>
<td>$Delivery block released$</td>
</tr>
<tr>
<td>BillingBlockReleased</td>
<td>$Billing block released$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event type</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateCreated</td>
<td>$SD.[Document type] = 'Return Order'$</td>
</tr>
<tr>
<td>PriceUpdated</td>
<td>$Price updated$</td>
</tr>
<tr>
<td>DeliveryBlockReleased</td>
<td>$Delivery block released$</td>
</tr>
<tr>
<td>BillingBlockReleased</td>
<td>$Billing block released$</td>
</tr>
</tbody>
</table>

Figure 3.4: Comparing the artifacts obtained using the XTract approach and our approach

To be able to identify multiple event types sharing the same timestamp column, similar cus-
tomization (as for the artifact identification) is also added because the original XTract approach which assigns a time stamp column entirely to one event types has its limitation. We illustrate this limitation with the DOCUMENT CHANGES table shown in Figure 1.1 which is a simplified version of the SAP change tables CDHDR and CDPOS. Three change records (with id 1, 2 and 3) of the sales order S1 are found, and one change record (with id 4) of the invoice B1 is found. The column Date changed can be automatically identified as one event type date changed using the XTract approach, which aggregates different changes. However, the table also shows different event types that are distinguished by the value in column change type. Since considering different changes as different event types reveals the relation between different change types in time aspect, it is required to be able to identify different event types based on the column Change type. We present two functions (similar to the two functions of artifacts creation): the first function createEventType($C_{time}$, $E_{name}$, $E_{condition}$) calls the existing XTract approach to obtain the event identifier and the event attributes which results in components that are sufficient for returning an event type; the second function createEventTypesByColumnValues($C'_{time}$, $C_e$) calls the first function createEventType($C_{time}$, $E_{name}$, $E_{condition}$) with $C_{time} = C'_{time}$, $E_{name} = v$, and $E_{condition} = \bigwedge_{i=1}^{n} C_i = v_i$, for each distinct tuple value $v = (v_1, \cdots, v_n)$ found for $C_e = (C_1, \cdots, C_n)$ (and similarly, if the columns in $C_e$ belongs to different tables than the time column $C_{time}$, tables can be selected and joined using the shortest path algorithm).

Identifying multiple event types is also illustrated in Figure 3.4. The artifact Sales Order shows the three different event types, PriceUpdated, DeliveryBlockReleased and BillingBlockReleased, identified from the time column Date Changed using the second function createEventTypesByColumnValues($C'_{time}$, $C_e$). In contrast, the XTract approach automatically considers the time column Date Changed as one event type ChangesChanged.

Furthermore, the approach presented allows users to change the name $A_{name}$ and condition $A_{condition}$ of an artifact and the event type names $E_{name}$ and event type conditions $E_{condition}$ of this artifact. It is also possible to change the event type identifier columns $C_{Eid}$ and the timestamp column $C_{time}$ of an event type. Moreover, users can easily delete and add event types, event type attributes and case attributes.

We conclude this chapter by giving an overview of the achievements shown in Figure 3.5. First, we reconstruct the data schemas from a given data source. Then, we identify artifact schemas and prevent the data divergence and convergence problem. Finally, we are able to semi automatically identify multiple artifacts from a given artifact schema.
Figure 3.5: Artifact Type Identification
Chapter 4

Interaction Identification

In Chapter 3, we have discussed the first step (A) the identification of complete stand-alone artifacts of the extraction phase, without considering any interactions between artifacts. In this chapter, we will introduce the second step, (B) interaction identification (between the artifacts), in which we are able to define interactions and discover both direct and indirect interactions between artifacts on the database level. Moreover, we will show our approach to (C) extract logs including interactions by extending the XTract approach. We first recall our current findings in the literature of interaction identification. Then, we introduce the four levels of interactions to clarify the concepts of interactions. Finally, we give an overview of our approach, in which the remaining sections of this chapter are also introduced.

No interaction identification techniques are discussed in the literature. Traditional log conversion approaches assume a single, isolated process, in which the notion of interactions with other processes are not considered. The XTract approach only has identified the artifacts without considering the interactions between the artifacts. Although the XTract approach can extract an event log based on an artifact, no information of interactions are included. Thus, unfortunately, due to the novelty of the notion of artifacts, no approach has yet been found in the literature which can automatically and specifically identify the interactions between artifacts (or between different event log conversion definitions). This also indicates that interactions between the artifacts have no clear definition or categorization yet. Moreover, the possibility of whether interactions can be identified on the database level has not been investigated. Therefore, we have conducted research into (1) defining interactions, and (2) designing a new approach to be able to automatically identify interactions and more importantly, (3) being able to extract event logs with interactions which can be used to discover process models that have interactions with each other.

Interactions between the artifacts should be discovered. For instance, take the SD table shown in Figure 1.1 and the artifact Return Order shown in Table 3.2. Without interactions, the Return Order artifact will only have two events, Date Created and Latest Change and have no reference at all to the related Return Order. This limitation leads to a great loss of important information, since the relation between artifacts do exist and are interesting to investigate. Furthermore, it is almost impossible to describe the behavior of an ERP system in a single process, whereas allowing to express interactions between processes also allows users to model the system as a collection of collaborative processes. In addition, an interaction between artifacts allows to express other relations than a one-to-one relation, which solves the data divergence and convergence problems.
To be clear about which interactions are discussed, the interactions are categorized into four levels based on the objects between which the interactions taken place. The categorization is shown in Figure 4.1. (a) The **artifact instance level interactions** denotes interactions between two **artifact instances** such as the sales order S1 and the return order S3. The existence of artifact instance level interactions between two artifacts also indicates, to some extent, the existence of an **artifact type level interaction** between the two artifacts. (b) The **artifact type level interactions** refer to interactions between two **artifacts** such as the **Sales Order** and the **Return Order**. (c) The **event level interactions** are interactions between two **events** such as the **Latest Change** event with id S1 and timestamp '1-6-2001' of the sales order S1 and the **Date Created** event with id S3 and timestamp '10-6-2001' of the return order S3. Similar to the artifact instance level interactions, the event level interactions between the two events also indicate, to some extent, the existence of an event type level interactions between the two event types of the two events. Thus, (d) the **event type level interactions** denote interactions between two **event types** such as the **Latest Change** event type of the sales order artifact and the **Date Created** event type of the return order artifact. For the sake of brevity, we also refer to artifact instance level interactions as instance level interactions and refer to artifact type level interactions as type level interactions.

In this chapter, we introduce a new approach to identify type level interactions and extract event logs with instance level interactions (instantiated from the type level interactions). In Chapter 5, we will consider event level and event type level interactions. An overview of our approach is shown in Figure 4.2. In Section 4.1, we first illustrate the concept of **type level interactions** and **instance level interactions** and give these two concepts clear formal definitions used in this approach, along with the reason why only these two types of interactions are identified on the database level. Then, in Section 4.2, a new method ‘type level interaction identification’ (1.1), which identifies interactions on the type level between **artifacts with conditions**, is described. The method ‘type level interaction identification’ also provides support to allow users to change ‘Artifact with type level interactions’ and select the interactions desired (1.2). In Section 4.3, we will illustrate the method ‘Mapping Creation’ which creates mappings for type level interactions (2.2), and extend the artifact mapping creation method provided by the XTract approach (2.1) to create a **mapping with type level interactions** for each artifact. Finally, based on a mapping with type level interactions, an artifact is converted to an event log with artifact instance level interactions by using and extending the conversion method provided by the XTract approach to be able to extract artifact instance level interactions.
interactions as attributes of traces (3).

4.1 Artifact Type Level and Instance Level Interactions

The aim of this section is to clarify the definition of type level and instance level interactions. We first define the direct interactions both on type level and instance level in Sections 4.1.1. Next, we extend the direct interactions to indirect interactions in Sections 4.1.2. Based on the definition and examples shown, we argue why only the type level and instance level interactions are identified on the database level, whereas the discovery of event type level and event level interactions is done on the log level in Section 4.1.3.

4.1.1 Direct Artifact Type Level and Instance Level Interactions

In this section, we first discuss how we use a reference between two main tables (of two artifacts) to define a direct interaction. Next, the formal definition of direct interactions and of an interaction graph is given.

To be able to automatically identify interactions between artifacts, the notion of interactions should be formally defined. Since a main table of an artifact contains the instances of this artifact, a reference $F$ found between two main tables may also indicate a direct relation between the two artifacts. For example, the reference $F_2$ between the main tables (SD and DD) of Sales Order and Delivery artifacts could be a direct interaction. But this combination of two artifacts and a reference between the main tables of these two artifact is not always a direct interaction. For example, the main tables of the artifacts Sales Order and Return Delivery is also related by reference $F_2$, but no real relations (i.e. no related instances) are found. This uncertainty is due to the fact that the instances of an artifact might only be a subset of the records in its main table $T_m$, which have no
CHAPTER 4. INTERACTION IDENTIFICATION

reference to the instances of other artifacts. In other words, even if a foreign key of a table exists in
the data structure, the foreign key might not be used by every record in this table.

To verify the existence of a direct relation between two artifacts inherited from a reference, there
should be a record of one artifact referring to a record in the main table of the other artifact. The
existence of such a direct relation between two artifacts is the concept of a type level interaction. The
definition of a direct type level interaction \( d = (A_S, F, A_T) \) \( \in A \times F \times A \) between two artifacts \( A_S \)
and \( A_T \) is a 3-tuple comprising the two artifacts and a reference \( F = (T_p, C_p, T_c, C_c, S_{condition}) \) \( \in F \),
where the main table \( A_S.T_m = T_p \) and the main table \( A_T.T_m = T_c \), thus the reference \( F = (A_S.T_m, \)
\( A_S.C_{Aid}, A_T.T_m, C_c, S_{condition}) \). We denote the artifact \( A_S \) as the parent artifact and the artifact
\( A_T \) as the child artifact. More importantly, there has to be a record \( s \) in \( A_S.T_m \) referring to a record \( t \)
in \( A_T.T_m \) that \( s \) satisfies the condition \( A_S.A_{condition} \), and \( t \) satisfy the condition \( A_T.A_{condition} \).
To be concrete, an example of relational algebra and an SQL query which select the number

\[
\begin{align*}
&\text{Listing 4.1: Interaction Count Selection Query} \\
&/* Select the count of a direct type level interaction } d = (A_S, F, A_T) */ \\
&/* In relational algebra: \\
&\#(\pi_{A_S.C_{Aid}, A_T.C_{Aid}}(A_S.T_m \bowtie_{(C_{p1}=C_{c1} \land \cdots \land C_{pn}=C_{cn} \land A_S.A_{condition} \land A_T.A_{condition})} A_T.T_m))) > 0 \\
&*/ \\
&\text{SELECT COUNT(DISTINCT } A_S.C_{Aid}, A_T.C_{Aid}) \\
&\text{FROM } A_S.T_M \text{ INNER JOIN } A_T.T_M \text{ ON } F.C_{p1} = F.C_{c1} \text{ AND } \ldots \text{ AND } F.C_{pn} = F.C_{cn} \text{ AND } F.S_{condition} \text{ IS TRUE AND } A_S.A_{condition} \text{ IS TRUE AND } A_T.A_{condition} \text{ IS TRUE} \\
\end{align*}
\]

\[
\begin{align*}
&\text{Listing 4.2: Interaction Instances Selection Query} \\
&/* Select the interaction instances of a type level interaction */ \\
&\text{SELECT DISTINCT } \text{SALESORDER.[SD ID]} \text{ AS [Sales Order SD id]}, \\
&\text{RETURNORDER.[SD ID]} \text{ AS [Return Order SD id]} \\
&\text{FROM } \text{SALESORDER.[SD ID]} = \text{RETURNORDER.[REFERENCE ID]} \\
&\text{AND SALESORDER.[DOCUMENT TYPE]} = 'Sales Order' \\
&\text{AND RETURNORDER.[DOCUMENT TYPE]} = 'Return Order' \\
\end{align*}
\]

We shall illustrate the direct type level and instance level interactions with the OTC example. A reference \( F_2 = \langle SD, \{SD id\}, DD, \{Reference SD id\} \rangle \) between the SD table and the DD table is found (i.e. the foreign key column Reference SD id in the DD table refers to the primary key column SD id of the SD table). Joining the two tables based on the reference \( F_2 \), the result shows two direct instance level interactions between the two delivery documents D1 and D2 and the sales order S1 and a direct instance level interaction instance between the return delivery D3 and the return order S3 is found, which indicate the existence of a direct type level interactions between the sales order and the delivery, and between the return order and the return delivery, respectively. However, there is no direct interactions found between the sales order and the return delivery, or
between the return order and the delivery. The result confirms our claim above: a reference between two artifacts does not always imply the existence of a direct type level interaction between the two artifacts. We obtained two direct type level interactions \((A_{SalesOrder}, F_2, A_{ReturnOrder})\) and \((A_{ReturnOrder}, F_2, A_{ReturnDelivery})\), respectively, with their direct instance level interactions shown in Tables 4.1 and 4.2.

An interaction graph for the OTC example shown in Figure 4.3 indicates the direct interactions (i.e. arcs) between the artifacts (i.e. elliptic nodes). Formally, we define an interaction graph \(G = (A_{sub}, D)\), where \(A_{sub} \subseteq A\) and \(D \subseteq A_{sub} \times F_{selected} \times A_{sub}\), where \(F_{selected} \subseteq F\). Each node \(A \in A_{sub}\) represents an unique artifact \(A\) (i.e. no two nodes represents the same artifact), and each edge \(e = (A_s, F, A_t) \in D\) represents the existence of a direct type level interaction \((A_s, F, A_t)\) between the parent artifact \(A_s\) and the child artifact \(A_t\), where \(F\) is the reference between the main tables of artifacts. The two subsets \(A_{sub}\) and \(F_{selected}\) is to allow a user to limit the scope of the interaction graph and interaction identification if desired. Note that this definition explicitly allows multiple edges (thus multiple direct type level interactions) between two artifacts because there could be different references linking the artifacts. Moreover, we define the outgoing edges of artifact \(A_s\) as \(\text{outEdges}(A_s) = \{(A_s, F_i, x) \in D \mid A_s, x \in A_{sub}\}\). Similarly, the incoming edges of artifact \(A_s\) is defined as \(\text{inEdges}(A_s) = \{(x, F_i, A_s) \in D \mid A_s, x \in A_{sub}\}\).

Table 4.1: Direct interactions \((A_{SalesOrder}, F_2, A_{ReturnOrder})\) instances

<table>
<thead>
<tr>
<th>Sales Order SD id</th>
<th>Return Order SD id</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S3</td>
</tr>
</tbody>
</table>

Table 4.2: Direct interactions \((A_{ReturnOrder}, F_2, A_{ReturnDelivery})\) instances

<table>
<thead>
<tr>
<th>Return Order SD id</th>
<th>Return Delivery DD id</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3</td>
<td>D4</td>
</tr>
</tbody>
</table>

Figure 4.3: The artifact type level interactions of the example.

Each eclipse underneath a table represents an artifact found in this table, and an arc represents the existing of a direct interaction between the two artifacts (i.e. there are reference instances).

4.1.2 Indirect Artifact level Interactions

In the previous section, we have defined direct interactions. In this section, we discuss the motivation for identifying indirect interactions. Then, we investigate different indirect interactions based on the direct interaction and define valid indirect interactions. Finally, we define the set of type
level interactions, which is constituted of the valid indirect interactions together with the direct interactions.

Identifying indirect interactions is required, since it should be possible to retrieve interactions between two artifacts when the interactions go through artifacts of which the logs will not be extracted. For instance, if the return orders shown in Figure 4.3 can be ignored, we would still like to be able to identify the interactions between the Sales Order and the Return Delivery. Another real-life situation encountered is that ERP systems in general have a document header and document lines structure. For example, the sales document table VBAK and the sales line table VBAP. Different types of documents are usually connected by lines. Typically, advisors are not interested in the lines but only at the header level. Since the document headers are connected through lines, it should be possible to identify interactions between header tables through the line tables, without extract any information of lines. However, it is still questionable whether each indirect interaction is possible and valid, e.g. the possibility of an indirect interaction between delivery and return order.

Indirect interactions can be defined using the direct interactions, similar to joining multiple tables linked by references. An important property which is used to determine the validness of indirect type level interaction is that, when given a reference, a record in the parent table is linked to zero or multiple records in the child table, whereas a record in the child table is only linked to zero or one records in the parent table. This property also indicates, given a direct type level interaction, an instance in the parent artifact can be linked to zero or multiple instances of the child artifact, whereas an instance in the child artifact can only be linked to zero or one instance of the parent artifact. We denote this property as Reference property. Similar to an inductive step, we can consider each possible case of an indirect interaction extended from two direct interactions and define this validness of this indirect type level interaction. Three possible cases of an indirect interaction are displayed in Figure 4.4.

The three possible cases of indirect interactions extended from two direct interactions

The first case (a) is a valid interaction, which is when two direct interactions \((A_{St_i}, F_i, A_{Ti})\) and \((A_{Ti_j}, F_j, A_{Tj})\) are linked as a chain and have the same direction, thus \(F_i.T_c = F_j.T_p\). For example, the two direct interactions \((A_{SalesOrder}, F_i, A_{ReturnOrder})\) and \((A_{ReturnOrder}, F_j, A_{ReturnDelivery})\) are linked this way. Since both interactions share the return order artifact, it is possible to join the three main tables based on the two direct interactions. As the references are joined in the same direction, we have the guarantee that an instance of the artifact \(A_{Ti_i}\) is only linked to zero or one
instance of the artifact $A_{S_i}$, based on the reference property. This property also indicates that an instance of the artifact $A_{S_i}$ is linked to all instances of the artifact $A_{T_i}$, and these instances of $A_{T_j}$ are not linked to any other instances of the artifact $A_{S_i}$. We denote this indirect type level interactions which is obtained from two direct interactions $(A_{S_i}, F_i, A_{T_i})$ and $(A_{T_j}, F_j, A_{T_j})$ of the same direction as a **strong indirect interaction** between the artifacts $A_{S_i}$ and $A_{T_j}$.

The second case (b) is a valid interaction but over-approximates the actual interactions, which is when two direct interactions $(A_{S_i}, F_i, A_{T_i})$ and $(A_{S_j}, F_j, A_{T_j})$ link to the same child artifact, thus $F_i F_j = F_j F_i$. For example, the two direct interactions $(A_{SalesOrder}, F_i, A_{Delivery})$ and $(A_{Invoice}, F_j, A_{Delivery})$ are linked this way. An instance of the artifact Sales Order can be explicitly linked to an instance of the artifact Invoice via one instance of the artifact Delivery, as shown by the graph in Figure 4.4(b) on the right-hand side (e.g. $(S1, B2), (S1, B2), (S2, B2)$). This explicitness is because an instance of the artifact Delivery can only be linked to one instance of Sales Order and one instance of Invoice (e.g. $(S1, D1, B1)$) based on the reference property. However, also based on the reference property, there is the possibility that an instance of the artifact Invoice are linked to multiple sales order (e.g. B2). As result, when the invoices B1 and B2 are selected as the indirect instance level interactions for the sales order S1, we have also included a part of B2 that is not related to S1 but related to S2. Therefore, all these type level interactions which contains two direct interactions $(A_{S_i}, F_i, A_{T_i})$ and $(A_{S_j}, F_j, A_{T_j})$ of the same child artifacts as a **weak indirect interaction** between the artifacts $A_{S_i}$ and $A_{S_j}$.

The third case (c) involves invalid indirect type level interactions which include two direct interactions $(A_{S_i}, F_i, A_{T_i})$ and $(A_{S_j}, F_j, A_{T_j})$ that have the same parent artifact $A_{S_i}$ and different target artifacts. For example, the two direct interactions $(A_{SalesOrder}, F_i, A_{Delivery})$ and $(A_{ReturnOrder}, F_j, A_{Delivery})$ are linked this way. Since an instance of the artifact Sales Order can be linked to multiple instances of the artifact Delivery and multiple instances of the Return Order (e.g. $(S1, \{D1, D2\}, \{S3\})$), it is impossible to determine the exact relations between the instances only based on the references. Therefore, we classify this case as invalid indirect type level interactions.

The validness of indirect interactions consisting of two direct interactions can be applied to the indirect interactions of any length. If an indirect interaction includes case (c), then it is an invalid interaction. Else if it includes a case (b), then it is a weak interaction which has a notion of over-approximation. Otherwise, it only consists of case (a), which is a strong interaction.

The definition of a valid type level interaction $I_{S,T} = \langle d_1, \ldots, d_n \rangle \in (\mathbb{A} \times \mathbb{F} \times \mathbb{A})^*$ between artifacts $A_S$ and $A_T$ is a sequence of direct interactions $d_i = (A_{P_i}, F_i, A_{C_i}) \in \mathbb{A} \times \mathbb{F} \times \mathbb{A}$ and $1 \leq i \leq n$, which satisfy one of the following property shown in Figure 4.5: (1) the parent artifact $A_{P_i}$ of first direct interaction $d_1$ is the artifact $A_S$, and the child artifact $A_{P_n}$ of the last direct interaction $d_n$ is the artifact $A_T$, and for $1 \leq i < n$, $A_{C_i} = A_{P_{i+1}}$; or (2) the child artifact $A_{C_i}$ of first direct interaction $d_1$ is the artifact $A_S$, and the parent artifact $A_{P_i}$ of the last direct interaction $d_n$ is the artifact $A_T$, and for $1 \leq i < n$, $A_{P_i} = A_{C_{i+1}}$; or (3) there is a number $k$ and $1 \leq k \leq n$, for $1 \leq i < k$, $A_{C_i} = A_{P_{i+1}}$, and $A_{C_k} = A_{C_{k+1}}$, and for $k < i < n$, $A_{P_i} = A_{C_{i+1}}$, and the parent artifact $A_{P_1}$ of first direct interaction $d_1$ is the artifact $A_S$, and the parent artifact $A_{P_n}$ of the last direct interaction $d_n$ is the artifact $A_T$. In addition, the number of interaction instances is greater than zero. The **length of strong joins** of this interaction is defined as $k$, and the **length of weak joins** is defined as $m = n - k$. When the length $m$ of weak joins is greater than zero, the interaction instances could be an over-approximation.
Given a set \( A \) of artifacts, a set \( D \) of direct interactions where for each \( d_i \in D \) the parent artifact \( A_{P_i} \in A \) and the child artifact \( A_{C_i} \in A \), and an artifact \( A_S \in A \), we define a full set \( \mathbb{I}_S^f \) of valid type level interactions of artifact \( A_S \) as \( \{ (I_{S,T}, A_T) \mid A_T \in A \text{ and } I_{S,T} \text{ is a valid type level interaction from } A_S \text{ to } A_T \} \). For example, the full set \( \mathbb{I}_{\text{ReturnOrder}}^f \) of the artifact Return Order is \( \{ (((\text{ReturnOrder}, F_2, \text{ReturnDelivery})), A_{\text{ReturnDelivery}}), (((\text{SalesOrder}, F_1, \text{ReturnOrder})), A_{\text{SalesOrder}}) \} \).

### 4.1.3 Possibility of Identifying Event Level Interactions

The definition and the examples have demonstrated that the type level interactions and instance level interactions between two artifacts are possible and relatively easy to be identified on the database level. In contrast, studying the OTC example has also revealed that to automatically identify the interactions between event types of two artifacts is much more difficult to do at the database level, since there might not be any explicit indication of a relation between two event types except the timestamps. For example, the direct interaction between the Sales Order and the Return Order does not indicate whether the Date Created event type of Return Order is (causally) related to the Date Created event type of Sales Order or to the Latest Change event type of the Sales Order. Similarly, the interaction on the event level, for which comparison in time might be needed, is also difficult to be identified in relational databases. For this purpose, the event log structure and process mining techniques are much more suitable. Therefore, we identified the type level interactions on databases and extract event logs with the instance level interactions. Then, we can use these event logs with instance level interactions to identify the interactions on event type level and event level.

### 4.2 Type Level Interaction Identification

In Section 4.1, we have introduced different kinds of interaction and formalized the type level interactions. In this section, we will present our approach to first automatically identify a set of type level interactions, and then allow users to select the type level interactions which they are interested in. A set of artifacts with type level interactions to each other are returned for log extraction.

Formally, given a set of artifacts \( A = \{ A_1, \cdots, A_n \} \) as input, where each artifact \( A = (A_{\text{name}}, C_{\text{Aid}}, E, C_{\text{attrs}}, I = \emptyset, S, A_{\text{condition}}) \), a number \( r > 0 \), a number \( k \geq 1 \) and a number \( m \geq 0 \), the algorithm should return the set \( A \) of artifacts, where for each artifact \( A_S \in A \), the set \( I_S \subseteq \mathbb{I}_S^f \) is filled, and for each interaction \( I_S \in I_S \), the length of strong joins is less than or equal to \( k \), the length
of weak joins is less than or equal to \( m \) and the number of distinct instance level of interactions is greater or equal to \( r \).

We made the following design decision explicitly. To restrict the identification of distinct interactions on the parent artifact, we constrain the number of strong joins greater than zero and the number of weak joins less than or equal to the number of strong joins, i.e. \( k \geq 1 \) and \( m \leq k \).

In other words, recalling the three situations of type level interaction shown in Figure 4.5, we only obtain interactions of the parent artifact \( A_S \) that satisfy (1), or (3) with \( m \leq k \). The interactions that satisfy (2) or (3) with \( m > k \) will only be extracted as the interactions of artifact \( A_T \). Note that this decision will also effect the event logs we extract. An instance level interaction will only be extracted from the parent artifact.

There are several advantages to only identify interactions on the one side. First, limiting the interactions to the parent artifact can prevent duplicated attributes to be extracted to improve the performance of log extraction. Furthermore, this decision also limits the number of discovered interactions to decrease the number of manual selections that are needed by users. Moreover, we obtain the parent and child structure and the reference property which can be reused during the process discovery. We have only observed one disadvantage. For example, when a child artifact instance has an interaction with a parent artifact instance but this parent instance is not extracted in the event log, then we miss the information that the child instance has an interaction in the event log. Note that we only restricted the numbers \( k \) and \( m \), thus it is very easy to change our solution to extract interactions on both side.

We divide the method into two parts. The first part takes a set of artifacts \( A = \{ A_1, \cdots, A_n \} \) as input, calculates a complete set of direct interactions between the artifacts, and returns an interaction graph \( G = (A_{sub}, D) \), where each node \( A_i \in A_{sub} \) represents an artifact \( A_i \), and each edge \( d = (A_s, F, A_t) \in D \) represents the existence of a direct interaction \( (A_s, F, A_t) \) between the artifact \( A_s \) and artifact \( A_t \). Having an interaction graph \( G \) created, the second part of this method use the interaction graph \( G \) as input and returns a set of artifacts, each of which includes a set of valid type level interactions \( I \) (direct and indirect).

### 4.2.1 Artifact Interaction Graph and Direct Interactions

In this section, we present our algorithm to construct an interaction graph \( G \) and calculate all direct interactions which satisfy the following criteria. Given a set of artifacts \( A = \{ A_1, \cdots, A_n \} \) as input, a complete set of direct interactions between the artifacts should be calculated, and an interaction graph \( G = (A_{sub}, D) \) should be returned, where each node \( A_i \in A_{sub} \) represents an artifact \( A_i \), and each edge \( d = (A_s, F, A_t) \in D \) represents the existence of a direct interaction \( (A_s, F, A_t) \) between the artifact \( A_s \) and artifact \( A_t \) where the number of instances of the direct interaction \( (A_s, F, A_t) \) should be greater than 0.

Each direct type level interaction with the number of instance level interactions greater than zero is added to the interaction graph, but this number does not have to be greater than or equal to \( r \). The reason for this is because a number of instance level interactions do not have a monotone decreasing property. For example, a sales order is linked to one return order, but this return order could be linked to hundred return deliveries. For \( r = 10 \), this interaction \( d \) between sales order and return order does not satisfy \( \text{countSelect}(d) \geq r \), but the indirect interaction \( d' \) between sales order and return delivery does satisfy \( \text{countSelect}(d') \geq r \). Since we use the interaction graph to compute
the indirect interactions, all direct interaction with the number of instance level interaction greater than zero should be added.

Algorithm ConstructInteractionGraph($A_{sub}$, $F_{selection}$)
1. $D \leftarrow \emptyset$
2. for $F = (T_p, C_p, T_c, C_c, S_{condition}) \in F_{selection}$
   3. do for $A_p \in Artifacts(T_p)$, $A_c \in Artifacts(T_c)$
      4. do if $countSelect(A_p, F, A_c) \geq 0$
         5. then $D \leftarrow D \cup (A_p, F, A_c)$
3. return $G = (A_{sub}, D)$

The algorithm ConstructInteractionGraph($A_{sub}$, $F_{selection}$) solves this problem. First, we initialize the set $D$. Second, for each reference $F \in F_{selection} \subseteq F$, we retrieve the artifacts $A_p$ of its parent table and the artifacts $A_c$ of its child table. We define the function $Artifacts(T) = \{A \in A_{sub} | A.T = T\}$ to retrieve the artifacts of which their main table is $T$ (see Line 3). For each combination of $A_p$ and $A_c$ with the reference $F$ between them, we select the count using the $countSelect$ query defined in Listing 4.1 to verify whether it is a direct interaction (see Lines 3 and 4). If the count is greater than 0, then an edge $(A_p, F, A_c)$ is added to $D$ as an edge in the interaction graph $G$.

Note that for a reference $F \in F \setminus F_{selection}$, this reference is not used to obtain any type level or instance level interactions. Another remark is that, when there is overlapping between the instances of artifacts in the same table, this overlapping is not considered as a direct interaction and no edge is added because the related reference is not given. Nevertheless, it is possible to add the references $F = (T_p, C_p, T_p, C_p, null)$ to the reference selection to solve this constraint.

4.2.2 Direct to Indirect Interactions

Having the interaction graph constructed, we present, in this section, our approach to identify type level interactions. The problem definition of the second part of the type level interaction identification is described as follows. Given a set $A$ of artifacts, an interaction graph $G = (A_{sub}, D)$, a number $r > 0$, a number $k \geq 1$ and a number $m \geq 0$, the algorithm should return, for each artifact $A_S \in A$, a set $I$ of valid interactions with other artifacts, where for each interaction $I \in I$, the number of interaction instances is greater than or equal to $r$, the length of strong joins is less than or equal to $k$ and the length of weak joins is less than or equal to $m$.

We present the algorithm CalculatesInteractions($A$, $G$, $r$, $k$, $m$), which solves this problem. First, for each artifact $A_S$, the algorithm initializes the set $I$ of interactions as empty set. For each outgoing edge $d = (A_S, F, A_t) \in D$ of $A_S$, the (direct) interaction $I_{current} = (d)$ is created. If the number of instance level interaction of $I_{current}$ is greater than or equal to $r$, then $I_{current}$ is added to the interaction set $I$ of artifact $A_S$. Next, algorithm calls the recursive algorithm calculateJoins($A_S$, $A_t$, $I_{current}$, $I_A$, $r$, $k - 1$, $m$) to select all valid indirect interactions.

Algorithm CalculatesInteractions($A$, $G$, $r$, $k$, $m$)
1. for $A_S \in A$
2. do $I \leftarrow \emptyset$
3. for $(A_S, F, A_t) \in outEdges(A)$
The algorithm \textit{calculateJoins} consists of the following steps. The input interaction \(I_{current}\) is a type level interaction between the source artifact \(A_S\) and the last artifact \(A_{current}\) added. If the length of strong joins of the current interaction \(I_{current}\) is still less than \(k\), i.e. \(k > 0\) (see Line 1), then for each outgoing edge \((A_{current}, F, A_{next})\) of \(A_{current}\), which is a direct interaction, we create a new interaction \(I_{next}\) from \(A_S\) to \(A_{next}\) by appending the new direct interaction \((A_{current}, F, A_{next})\) to \(I_{current}\) (see Line 3). Using a similar query as Listing 4.1, we calculated the count of the new interaction \(I_{next}\). If \(count \geq r\), \(I_{next}\) is added to the set \(\mathcal{I}\) of interactions of \(A_S\) (see Line 6). Then, if \(count > 0\), then there is a chance to find an indirect interactions with \(count \geq r\), thus the algorithm \textit{calculateJoins} calls itself with the next artifact \(A_{next}\), the new interaction \(I_{next}\), and the number of strong length is \(k - 1\) (see Line 8).

\begin{algorithm}
\caption{\textit{calculateJoins}(\(A_S\), \(A_{current}\), \(I_{current}\), \(\mathcal{I}\), \(r\), \(k\), \(m\))}
\begin{algorithmic}[1]
\State \textbf{if} \(k > 0\)
\State \textbf{then for} \((A_{current}, F, A_{next}) \in \text{outEdges}(A_{current})\)
\State \hspace{1em} \textbf{do} \(I_{next} \leftarrow I_{current} \cup \{(A_{current}, F, A_{next})\}\)
\State \hspace{1em} \textbf{count} \leftarrow \text{selectCount}(I_{next})
\State \hspace{1em} \textbf{if} \(\text{count} \geq r\)
\State \hspace{2em} \textbf{then} \(\mathcal{I} \leftarrow \mathcal{I} \cup I_{next}\)
\State \hspace{1em} \textbf{if} \(\text{count} > 0\)
\State \hspace{2em} \textbf{then} \textit{calculateJoins}(\(A_S\), \(A_{next}\), \(I_{next}\), \(\mathcal{I}\), \(r\), \(k - 1\), \(m\))
\State \textbf{if} \(m > 0\)
\State \textbf{then for} \((A_{next}, F, A_{current}) \in \text{inEdges}(A_{current}) \setminus \text{set}(I_{current})\)
\State \hspace{1em} \textbf{do} \(I_{next} \leftarrow I_{current} \cup \{(A_{next}, F, A_{current})\}\)
\State \hspace{1em} \textbf{count} \leftarrow \text{selectCount}(I_{next})
\State \hspace{1em} \textbf{if} \(\text{count} \geq r\)
\State \hspace{2em} \textbf{then} \(\mathcal{I} \leftarrow \mathcal{I} \cup I_{next}\)
\State \hspace{1em} \textbf{if} \(\text{count} > 0\)
\State \hspace{2em} \textbf{then} \textit{calculateJoins}(\(A_S\), \(A_{next}\), \(I_{next}\), \(\mathcal{I}\), \(r\), \(0\), \(m - 1\))
\State \textbf{return}
\end{algorithmic}
\end{algorithm}

For the current artifact \(A_{current}\), after all outgoing edges are processed, the algorithm \textit{calculateJoins} verifies whether the length of weak join is still less than \(m\), i.e. \(m > 0\). If true, for each incoming edge \((A_{next}, F, A_{current})\) of \(A_{current}\), which is a direct interaction, the algorithm creates a new current interaction \(I_{next}\) by appending the new direct interaction \((A_{next}, F, A_{current})\) to \(I_{current}\) (see Line 11). If the number of instance level interactions of \(I_{next}\) is greater than or equal to \(r\), i.e. \(count \geq r\), then the new interaction \(I_{next}\) is added to \(\mathcal{I}\) (see Lines 13 and 14). If \(count > 0\), then we call the recursive call with the new artifact \(A_{next}\) and the new interaction \(I_{next}\). After appending a weak join, no strong joins is allowed to be appended to the interaction, since that violates with the definition of a valid interaction. Therefore, the length of strong join is set to 0 to avoid recursive calls appending strong joins. We define \(\text{set}(I) = \{d_i \mid d_i \in I\}\) to retrieve the set of direct interactions of
CHAPTER 4. INTERACTION IDENTIFICATION

a type level interaction $I$, where $I = \langle d_1, \ldots, d_n \rangle$. To avoid the weak join taking the same direct
interactions and creating too complex weak join, we remove the direct interactions that is already
in the current interaction $I_{\text{current}}$ (see Line 10).

After the algorithm \texttt{CalculatesInteractions}(A, G, r, k, m) terminates, we have, for each artifact
$A \in \mathbb{A}$, the set $A.I$ of interactions calculated and returned. Users can now select a set of desired
type level interactions from the set $A.I$.

4.3 Mapping Creation and Log Extraction

In Section 4.2, we have presented our approach for identifying type level interactions which returns
a set of artifacts, each of which has a set of type level interactions to other artifacts. In this section,
given an artifact with type level interactions to other artifacts, we would like to (C) extract an
event log for the artifact including the instance level interactions. We first give a short problem
description of extracting event logs of artifacts. Then, we present an overview of our solution to
the log extraction (C) problem. Next, we highlight the extensions that is added to the mapping
structure. Finally, an overview of this chapter is given.

We have argued that it is more logical to identify the interactions between events and between
event types on the event log level. Therefore, the type level interactions and instance level inter-
actions should be extracted in the event log of an artifact. The XTract approach already provided
methods which create a mapping (also known as conversion definition) between an artifact and an
event log structure definition and use the mapping to convert artifact data to an event log. However,
no mapping definition existed for the type level interactions yet. Furthermore, the new definition of
artifacts and event types, which are extended with conditions, are not supported by the mapping
structure of XTract. The aim is thus to extend the mapping structure of the XTract approach such
that a mapping can be created for the extended artifact with type level interactions. Moreover, an
event log with instance level interactions can be extracted from the artifact based on the mapping.

The solution proposed for (C) the LogExtraction follows the steps done by the original XTract
approach and is demonstrated in Figure 4.6. First, an artifact (e.g. artifact sales order shown in
Figure 4.6) is used to create a log mapping. The type level interactions are mapped as a ListAttribute
where the links are direct interactions (e.g. the LogMapping of sales order is shown in Figures 4.6
and B.1). Having the log mapping created, the new ‘LogExtraction’ method extends the XTract
approach to be able to append conditions to the queries and use direct interactions in the joins.
Using the new queries, the cache database is created and the data structure of the cache database
is unchanged from the XTract approach. The interactions are stored as attributes of traces.

The new mapping structure is illustrated in Figure B.2, where the uncolored classes are already
defined by the XTract approach. A class GeneralMappingProperty as an attribute of the GeneralMappingItem,
which contains two attributes, a condition and the artifact identifier, is added. By
adding this class, the condition attribute of the extended definition of artifact and event types can
be mapped to the condition of the GeneralMappingProperty. Moreover, the ListAttribute is used to
map interactions. An inheritance class DirectInteraction is added to the class Reference to be used
by ListAttribute to specify interactions. This way a mapping for the interaction can be created as
listAttribute: key is ‘interaction_’ concatenated with the interacting artifact name because XES does
not allow the same key to be used multiple times. The attributeId is the set $A_i.C.Aid$ of identifier
columns. The from table is the main table $A_s.T_m$ of the source artifact. The link is equal to the interaction.

An important design decision we made is that, from an artifact to each of the other artifacts, we only allow one type level interactions to be extracted between these two artifacts. This is mainly due to the ambiguity when mining different type level interactions between two artifacts in the mining phase (II). We shall discuss this more in detail in Section 5.1.1.

We conclude this chapter by giving an overview of the achievements shown in Figure 3.5. In Section 4.2, we have shown how to compute, for each of the artifacts obtained in Chapter 3, a set of type level interactions from itself to other artifacts. In this section, we have discussed that for each of these complete artifacts such as the Sales Order artifact shown in Figure 3.5 on the left hand side, we create a log mapping including information about type level interactions. Next, we translate the log mapping to SQL queries to build a cache database to store the event log data. Instance level interactions are retrieved by queries as case attributes. Finally, using OpenXes, we write an event log from the case database. Now, we have a list of event logs with instance level interactions.
Figure 4.6: Mapping Creation and Log Extraction

An artifact is used to create a log mapping, which is used to create a cache database, which is used to create the event log.
Chapter 5

Artifact-Centric Process Discovery and Analyses

In Chapter 4, we have presented our approach to extract event logs with artifact instance level interactions for artifacts with type level interactions. In this chapter, we illustrate how the logs are used to identify life cycles for each artifact and the interactions between the event types of artifacts. Moreover, we present our method to create a proclet system given these life cycles and event type level interactions and visualizing the proclet system using a simple representation which can be more easily explained to business users. Finally, we provide several analyses to allow users to assess the process discovered. But first, we discuss the motivation for event type level interaction identification and give an overview of our approach.

No previous work has been found on (semi-)automated interactions discovery for event logs. Classic process discovery algorithms assume a single event log as input and return a discovered process model as output. Since no interactions have been identified for the artifacts and the event logs by the original XTract approach, the XTract approach has also limited its scope to only apply existing discovery algorithms and return a single, isolated control flow model (also referred to as life-cycle) per artifact. As result of the previous chapter, a set of event logs with trace level interactions (i.e. artifact instance level interaction) are available, which indicate the possibility of identifying event level interactions and event type level interactions. We argue that being able to identify event type level interactions is one of the most valuable steps of the artifact-oriented approach, since the artifact type level and artifact instance level interactions might already be known and relatively easy to identify. In contrast, the interactions on event type level not only have the implication of much detailed dependencies but also new information for users which is not easy to detect using database technology or data mining technology. Furthermore, due to the business context of this thesis, the discovered model should be intuitive and understandable. Additional information should be provided for business users to analyze the discovered model.

We have formulated the problem as follows. Given a set of event logs with trace level interactions (i.e. artifact instance level interaction), the approach should return a process model describing the life cycle of each log as well as the interactions between the life cycles. Moreover, the returned model should be informative and understandable. Figure 5.1 shows an overview of the approach. Starting from the logs with interactions, the existing process discovery miners are used to discover a life-cycle for each event log (1), which is described in Section 5.1.2. To identify event type level
interactions, two methods are proposed and presented in Section 5.2. The first method merges the logs and applies discovery miners to extract interactions (2), whereas the second method defines a criterion of a valid event type level interaction and finds the event type level interactions based on the criterion (3). The retrieved event type level interactions are added to the life-cycle models to create a proclet model (4). Using the proclet model as a formal underlying data model, a simple representation and more detailed information are visualized for business users. Before describing the event type level interaction identification in detail, we would like to first clarify the input, event logs with trace level interactions, and briefly introduce the discovery of life-cycles of artifacts.

Figure 5.1: Method Artifact-Centric Process Discovery and Analysis
Each rectangle represent an object, and each arc represents a method used.

5.1 Logs and Life-cycle Discovery

In this section, we shall first introduce the concept of event logs with trace level interactions and define several functions used to retrieve interactions. Then, we illustrate the discovery of life-cycles from a given event log using the existing process discovery techniques.

5.1.1 Event logs with trace level interactions

In Section 2.1.1, we have defined an event log as a set of cases $L \subseteq \mathcal{L}$, where $\#_{\text{trace}}(c) = \sigma_c \in \mathcal{E}^*$ returns the trace of events of this case to be able to reason about event logs. A list of definitions regarding trace level interactions are defined in this section and used in this chapter. In Chapter 4.3, we have illustrated how to extract artifact instance level interactions of an artifact as attributes of the traces. Therefore, given an event log $L$ and another log $T$, we can use $I_I(c) \in AN : \#I_I(c)$ to retrieve the set of cases of log $T$ to which the case $c \in L$ has an interaction with, where the artifact of the log $L$ is the parent artifact. We recall an important design decision discussed in Section 4.2: the interactions between two artifacts are only extracted for the parent artifact to avoid extract duplicate attribute and improve the performance of log extraction. Thus, the definitions and functions given below have taken this decision into account.
We define the functions $I$, $I_P$ and $I_C$ to retrieve the trace level interactions. We define $I(L_S, L_T) = \{(c_s, c_t) \mid c_s \in L_S \land c_t \in L_T \land (c_s, c_t) \in \#I_T(c_s)\} \subseteq L_S \times L_T$ as the combination oriented set of trace level interactions between the logs $L_S$ and $L_T$, where each $(c_s, c_t) \in I(L_S, L_T)$ indicates that there is a trace level interaction between the two traces $c_s$ and $c_t$. Note that the condition '$c_t \in L_T$' in the definition indicates that the combination oriented set of trace level interactions $I(L_S, L_T)$ is different than the set of all interaction case attributes of log $L_S$. The difference is due to the possibility that a case of artifact $T$, to which a case of artifact $S$ has interaction with, is not extracted in the log $L_T$ (e.g., not in the extraction time frame). If one has extracted the trace level interactions between two artifacts on both artifacts, then the condition '$c_s \in \#I_S(c_t)$' should be included in the definition.

In addition, we define a parent oriented set of trace level interactions as follows. $I_P(L_S, L_T) = \{(c_s, \{c_t \mid c_t \in L_T \land (c_s, c_t) \in I(L_S, L_T)\}) \subseteq L_S \times P(L_T)$ between the logs $L_S$ and $L_T$, which is simply group the $I(L_S, L_T)$ by parent traces $c_s$. We define a child oriented set of trace level interactions as follows. $I_C(L_S, L_T) = \{(c_s, \{c_t \mid c_t \in L_T \land \forall 1 \leq n (c_s \in L_S \land (c_s, c_t) \in I(L(S, L_T)))\} \subseteq P(L_S) \times L_T$ between the logs $L_S$ and $L_T$.

We illustrate the functions $I$, $I_P$ and $I_C$ with the OTC example shown in Figure 4.4(b). Assume we have extracted an event log of the artifact $SalesOrder$ with trace level interactions to the artifact $Invoice$ and an event log of the artifact $Invoice$ (i.e., the artifact $SalesOrder$ is the parent artifact). We use the trace id to represent the trace. Thus the combination oriented set of trace level interactions $I(L_{SalesOrder}, L_{Invoice}) = \{(S1, B1), (S1, B2), (S2, B2)\}$. The parent oriented set of trace level interactions $I_P(L_{SalesOrder}, L_{Invoice}) = \{(S1, \{B1, B2\}), (S2, \{B2\})\}$. The child oriented set of trace level interactions $I_C(L_{SalesOrder}, L_{Invoice}) = \{(\{S1\}, B1), (\{S1, S2\}, B2)\}.

Note that we assume that the trace level interactions are only based on one artifact type level interaction between two different event logs $L_S$ and $L_T$ (also discussed in Section 4.3). In other words, an event log $L_S$ has only one attribute with the name $I.T \in AN$. This constraint is due to that the event type level interaction identification are based on $I(L_S, L_T)$. When there is more than one artifact type level interaction, it is difficult to decide which one of the artifact type level interactions should be used, or if multiple artifact type level interactions should be used. Due to the time scope, we left the possibility of identifying event type level interactions based on multiple artifact type level interactions as future work.

To be able to filter a trace of log $L$ such that, given an even type $E \in A_L$, only events with event type $E$ are retained, we define the project function $F_E$. For example, the trace sales order $\sigma_{S1}$ consists of two events, i.e., $((S1, created, 13 - 5 - 2001), (S1, latestchange, 1 - 6 - 2001))$. If we apply the project function on the trace $\sigma_{S1}$ with $E = created$, then we will have $F_{created}(\sigma_{S1}) = ((S1, created, 13 - 5 - 2001))$.

For later use, we define the following notations. The symbol $\sigma_s <_T \sigma_t$ between two sequences $\sigma_s$ and $\sigma_t$ means that for each event $e_s \in \sigma_s$, its timestamp $\#time(e_s)$ is before the timestamps $\#time(e_t)$ of each event $e_t \in \sigma_t$. For example, $F_E(\sigma_s) <_T F_E(\sigma_t)$ means that each event $e_s \in \sigma_s$ which has the event type $E$ is executed earlier than each event $e_t \in \sigma_t$ which has the event type $E$. In the same way, the symbol $\sigma_s =_T \sigma_t$ denotes that all events of the two traces happened on the same time, and the symbol $\sigma_s >_T \sigma_t$ is the opposite of $\sigma_s <_T \sigma_t$, which means the same as $\sigma_t <_T \sigma_s$. In addition, $\sigma_s \leq_T \sigma_t$ and $\sigma_s \geq_T \sigma_t$ are also used.
5.1.2 Life-cycle Miners

In this section, we give a general definition of the algorithms that discover the life cycle of an artifact from a given event log. Many existing process discovery algorithms can already automatically discover a control flow model when given an event log. Within this thesis, the control flow model is considered to be the life-cycle of the given event log which described the behavior of an artifact. Therefore, we can simply use the existing discovery techniques. We generally define the life-cycle discovery problem, without discovering interfaces and channels, as follows. Given an event log \( L \), the algorithm returns a petri net model \( PN = (P, T, F, T_v) \) describing the life-cycle of the log \( L \), and we use the term life-cycle miner \( \text{Miner}(L) \) to denote any such a process discovery algorithm. We can reuse any existing process discovery algorithm that returns a Petri Net as implementations of the life-cycle miner \( \text{Miner} \). For instance, Alpha miner \([5]\), Genetic miner \([18]\), and ILP miner \([31]\) can be used directly. The Flexible Heuristic miner \([30]\) returns a heuristic net which can be converted to a petri net using the HeuristicsNetToPetriNetConverter plug-in provided in the same HeuristicMiner package. Additionally, the ProM package Murata\(^1\) provided by ProM can be used to reduce the silent transitions in the returned petri net.

5.2 Event Type Level Interaction Identification

In Section 5.1.2, we have shown how to identify a life cycle for each event log. In this section, we illustrate two new methods that are proposed to identify interactions between the event types of event logs. More formally, for each two event logs \( L_S \) and \( L_T \) in the given set of logs, we would like to identify the set \( X \subseteq (A_{L_S} \times A_{L_T}) \cup (A_{L_T} \times A_{L_S}) \) of event type level interactions, where \( A_{L_S} \) and \( A_{L_T} \) are the sets of all event types appearing in logs \( L_S \) and \( L_T \), respectively. Note that this definition only defines the form of event type level interactions, but does not give any meanings to the event type level interactions, since the meaning of event type level interactions varies depending on the method we deploy to discover them.

5.2.1 Interaction Discovery by Merging Logs

In this section, we propose our first method to discover event type level interactions, which is merging the logs based on trace level interactions. We apply a process discovery miner on the merged log to identify dependencies between event types of the two artifacts, which can be translated as an event type level interaction.

We present the \text{CalculateETLInteractionsByMergingLogs} algorithm below. First in Lines 1 and 2, if there is no trace level interactions found between the logs \( L_S \) and \( L_T \), i.e. \( I(L_S, L_T) = \emptyset \), then the algorithm returns an empty set as the event type level interactions. Else, for each \( (\sigma_s, \sigma_t) \in I(L_S, L_T) \), we merge the two traces to a new trace using the merge function \( \mathcal{M} \) (see Line 5).

The merge function \( \mathcal{M} \in (\mathcal{E}_S^* \times \mathcal{E}_T^*) \rightarrow (\mathcal{E}_S \cup \mathcal{E}_T)^* \) merges the events of two input traces \( \sigma_s \) and \( \sigma_t \) recursively and ordered by the timestamps: (a) \( \sigma_s = \langle \rangle \) and \( \sigma_t = \langle \rangle \), \( \mathcal{M}(\langle \rangle, \langle \rangle) = \langle \rangle \). (b) For \( \sigma_s \in \mathcal{E}_S^* \) and \( \sigma_t = \langle \rangle \), \( \mathcal{M}(\sigma_s, \langle \rangle) = \sigma_s \). (c) For \( \sigma_t \in \mathcal{E}_T^* \) and \( \sigma_s = \langle \rangle \), \( \mathcal{M}(\langle \rangle, \sigma_t) = \sigma_t \). (d) For

---

\(^1\)https://svn.win.tue.nl/trac/prom/browser/Documentation/PackageMurata.pdf
CHAPTER 5. ARTIFACT-CENTRIC PROCESS DISCOVERY AND ANALYSES

\[ \sigma_s \in E_{\mathcal{S}}^s \land s \in E_{\mathcal{S}}, \text{ and for } \sigma_t \in E_{\mathcal{T}}^t \land t \in E_{\mathcal{T}}: \]

\[ \mathcal{M}(\langle s \rangle + \sigma_s, \langle t \rangle + \sigma_t) = \begin{cases} \langle s \rangle + \mathcal{M}(\sigma_s, \langle t \rangle + \sigma_t) & \text{if } T(s) \leq T(t) \\ \langle t \rangle + \mathcal{M}(\langle s \rangle + \sigma_s, \sigma_t) & \text{if } T(s) > T(t) \end{cases} \]

The merged log \( L_{\text{new}} \) from the trace level interactions \( I(L_S, L_T) \) consists of all merged traces \( \sigma_{\text{new}} \), of which the number of events is greater than zero (see Line 6). Now, a process discovery algorithm can be applied on the merged log \( L_{\text{new}} \) to discover dependencies between the two sets \( A_{L_S} \) and \( A_{L_T} \) of event types. We use \( \text{Miner}(L) \) to denote a process mining algorithm which takes a (merged) log and returns a new process model \( PN_{\text{new}} \) (see Line 7). For each direct succession \((E_i, E_j)\) between the two event types \( E_i \) and \( E_j \) indicated by the model \( PN_{\text{new}} \), we verify whether the direct succession satisfies the event type level interaction form. Thus, if the two event types belong to different original sets \( A_{L_S} \) and \( A_{L_T} \), then this direct succession is considered to be an event type level interaction between \( E_i \) and \( E_j \) and added to the set \( X \) (see Lines 8-10).

**Algorithm** CalculatesETLInteractionsByMergingLogs\((L_S, L_T)\)

1. **if** \( I(L_S, L_T) = \emptyset \) **then return** \( \emptyset \)
2. **else** \( L_{\text{new}} \leftarrow \emptyset, X \leftarrow \emptyset \)
3. **for** \((\sigma_s, \sigma_t) \in I(L_S, L_T)\) **do** \( \sigma_{\text{new}} \leftarrow \mathcal{M}(\sigma_s, \sigma_t) \)
4. **if** \( \sigma_{\text{new}} \neq \emptyset \) **then add** \( \sigma_{\text{new}} \) **to** \( L_{\text{new}} \)
5. \( PN_{\text{new}} \leftarrow \text{Miner}(L_{\text{new}}) \)
6. **for** \((E_i, E_j)\) **where** \( E_j \) **is a direct successor of** \( E_i \) **in the model** \( PN_{\text{new}} \)
7. **and** \((E_i, E_j) \in A_{L_S} \times A_{L_T} \) **or** \((E_i, E_j) \in A_{L_T} \times A_{L_S} \)
8. **do** add \((E_i, E_j)\) **to** \( X \).
9. **return** \( (L_{\text{new}}, X) \)

We illustrate here an example of how to translate direct successors as event type level interactions. For instance, if we use the heuristic miner as the \( \text{Miner}(L_{\text{new}}) \), a simple causality net \( DG = (A_{L_{\text{new}}}, E) \), in which \( E \subseteq A_{L_{\text{new}}} \times A_{L_{\text{new}}} \) indicates the direct succession between event types, is obtained from the log \( L_{\text{new}} \). Since this log \( L_{\text{new}} \) is merged from the two logs \( L_S \) and \( L_T \), i.e. \( A_{L_{\text{new}}} = A_{L_S} \cup A_{L_T} \), which implies that \( E \subseteq (A_{L_S} \cup A_{L_T}) \times (A_{L_S} \cup A_{L_T}) \). Thus, a dependency \((s, t) \in E \) and more importantly \((s, t) \in (A_{L_S} \times A_{L_T}) \cup (A_{L_T} \times A_{L_S}) \) is returned as an event type level interaction because \((s, t) \in (A_{L_S} \times A_{L_T}) \cup (A_{L_T} \times A_{L_S}) \) \( \Leftrightarrow ((s \in A_{L_S} \land t \in A_{L_T}) \lor (t \in A_{L_S} \land s \in A_{L_T})) \), thus satisfying the form of an event type level interaction. Similarly, if we used a miner which returns a labeled Petri net \( PN = (P, T, F, T_v) \) as \( \text{Miner}(L) \), then we can construct a simple causality net \( DG \) to obtain direct successors and event type level interactions.

An important remark is that different process discovery techniques return a different set of event type level interactions. The meaning of an event type level interactions identified also varies depending on the discovery miner chosen as \( \text{Miner}(L) \). For example, the interactions (dependencies) between two event types identified by the alpha miner are absolute precedence, thus event type A always before event type B, and no B is found before A in the log. In contrast, the event-type level interactions returned by the flexible heuristic miner have a different meaning, i.e. event type A is mainly before event type B, and B before A is much less frequent (or lower than the threshold) in the log.
5.2.2 Interaction Discovery by using Criteria

In Section 5.2.1, we have shown how to merge the event logs and apply miners on the merged log to discover event-type level interactions. But no event level interactions were identified. Moreover, the discovery miners generally only indentify the main flows, thus the main event-type level interactions, whereas the analysts are interested in outliers. Therefore, in this section, we propose the second method to discover a set of event-type level interactions between two event logs based on one of the predefined criteria. The method can be applied with or without merging the logs. We first briefly describe our motivation, followed by a list of criteria we defined, in which we also discussed our approach to identify all possible event type level interactions and unusual event type level interactions based on event level interactions.

The reason why we also used the criteria method is because the event type level interactions discovered by merging traces vary in meaning. For analysts, it might be difficult to understand and assess event-type level interactions based on the meaning implicitly defined by these miners. Another reason is that when two sets of event types tends to be parallel, many event type level interactions between the two sets are identified, which leads to a rather complex artifact centric model which is difficult to analyze. This complexity blurs analysts’ focus and obstructs analysts from identifying the unusual flows. More importantly, analysts might want to focus on particular criteria (e.g. an event type level interaction with shortest average duration) or focus on particular set of event types, while existing miners only discover the model defined by these miners. Our last argument is that the discovery miners generally only indentify the main flows and filter out the outliers, while interviews conducted with advisors indicate that they are more interested in the outliers. Thus, we would also like to be able to obtain all possible event type level interactions, or at least all possible direct succession relations of event types.

We define several criteria, and for each criterion, a suitable situation is given as an example to use this criterion. First, we define the general context as given two logs $L_S, L_T$, with two sets $A_S, A_T$ of event types, respectively, and $|I(L_S, L_T)| > 0$. Users have selected the sets of event types $A_{subS}, A_{subT}$, with $A_{subS} \subseteq A_S, A_{subT} \subseteq A_T$. We would like to find a set $X$ of event type level interactions, where each $(E_p, E_c) \in X$ satisfies the selected criterion and $X \subseteq (A_{subS} \times A_{subT}) \cup (A_{subT} \times A_{subS})$.

First, for instance, we have two artifacts, each of which consists of many changes. It might be interesting to identify which changes happen always before the other changes without exceptions while other changes could happen between. For this purpose, we define absolute precedence criterion as follows. For each $(E_p, E_c) \in A_{subS} \times A_{subT} \cup A_{subT} \times A_{subS}$:

$$(E_p, E_c) \in X \iff \text{for all } (\sigma_s, \sigma_t) \in I(L_S, L_T) : F_{E_p}(\sigma_s) \leq_{T} F_{E_c}(\sigma_t)$$

It is possible that there might be exceptions. For example, the event type Created of sales orders should always happened before the event type Created of the related deliveries, but the recorded data may have been entered manually and hence it may contain mistakes, i.e. some creations of deliveries happened before the creation of orders (and other events could happened between). Therefore, to be able to detect both the main flow and the exceptions, we define existing precedence criterion as follows. For each $(E_p, E_c) \in A_{subS} \times A_{subT} \cup A_{subT} \times A_{subS}$:

$$(E_p, E_c) \in X \iff \text{there is } (\sigma_s, \sigma_t) \in I(L_S, L_T) : F_{E_p}(\sigma_s) <_{T} F_{E_c}(\sigma_t)$$
We might say that when one event type always happens shortly after another event type, i.e. the time duration between the events of the two event types is shortest, then there might be a causal relation. We define this as another criterion shortest time which select only two event type level interactions:

\[
(E_p, E_c) \in X \iff \min_{(E_p,E_c) \in A_{subT} \times A_{subS}} \left( \frac{\sum_{(s_{x},s_{y}) \in I(L_S,L_T)} \text{AvgTimeDur}(F_{E_p}(s_x),F_{E_c}(s_y))}{|I(L_S,L_T)|} \right) \leq \min_{(E_p,E_c) \in A_{subT} \times A_{subS}} \left( \frac{\sum_{(s_{x},s_{y}) \in I(L_S,L_T)} \text{AvgTimeDur}(F_{E_p}(s_x),F_{E_c}(s_y))}{|I(L_S,L_T)|} \right)
\]

Note, for the above three criteria, we could also used the parent oriented interactions set \(I_P(L_S,L_T)\) or the child oriented interactions set \(I_C(L_S,L_T)\). However, when the merged log is used, the criteria should based on the combination oriented interactions.

**Event Level Interaction and Outlier Identification**

We have argued that discovery miners only identify main flows (i.e. only main event type level interactions), and analysts would like to identify outliers. Therefore, we propose a criterion using the merged log to identify all possible event-type level interactions based on event level interactions. First we define an event level interaction as follows. For a direct succession of two events \((e_i, e_{i+1})\) in a merged trace, if they belong to different original event sets (of the two log), then there is an event level interaction between them. Formally, if \((e_i, e_{i+1}) \in M(\sigma_s, \sigma_t) \land (\sigma_s, \sigma_t) \in I(L_S,L_T) \land ((e_i \in E_S \land e_{i+1} \in E_T) \lor (e_i \in E_T \land e_{i+1} \in E_S))\), then \((e_i, e_{i+1})\) is an event level interaction.

An important remark here is that the event level interactions we defined is totally unrelated to the set \(X_{\text{minerOnMergedLog}}\) of event type level interactions we identified by applying a miner on the merged log. Thus, event level interactions (defined here) are not instantiated from these event type level interactions. However, the merged log can be used to identify these event level interactions between the logs \(L_S\) and \(L_T\). Moreover, we can use these event level interactions, to identify all possible combination of a direct succession of two event types, for which an event level interaction existed. We define this condition as the existence of an event level interactions criterion:

\[
(E_p, E_c) \in X \iff \exists (\sigma_s, \sigma_t) \in I(L_S,L_T) : \exists (e_i, e_{i+1}) \in M(\sigma_s, \sigma_t) : #\text{eventType}(e_i) = E_p \land #\text{eventType}(e_{i+1}) = E_c \land ((e_i \in E_S \land e_{i+1} \in E_T) \lor (e_i \in E_T \land e_{i+1} \in E_S))
\]

We can use the set \(X_{\text{existenceEventLevelI}}\) of event type level interactions identified based on the the existence of an event level interactions criterion to illustrate the possible event type level interactions (i.e. possible combination of direct succession of two event types) that is not discovered by applying a miner on the merged log. Since we assume the miners identify the main flows, we consider \(X_{\text{existenceEventLevelI}} \setminus X_{\text{minerOnMergedLog}}\) as the set of unusual event type level interactions (i.e. outliers).

To only retrieve the event type level interaction in \(X_{\text{existenceEventLevelI}}\) that has the maximal number of event level interactions found in the merged log, we define the last criterion max number of event level interactions as:
For this thesis, we have only defined and implemented the aforementioned criteria and used these in the case studies. Investigating and implementing other criteria is listed as future work.

5.2.3 Limitations

One of the main limitations of these event type level interaction identification is that the interactions are limited to two artifacts only. For example, we have three event logs, sales order, deliveries, and invoices. The sales orders have interactions towards deliveries, and the deliveries have interactions towards invoices. When we have identified the creation of invoices has lead to the creation of deliveries, and the creation of orders has also lead to the creation of deliveries, but no information relating the creation of orders and the creation of invoices can be obtained.

For the above mentioned limitation, two solutions can be applied. One can identify the artifact type level interactions between the sales orders and invoices, and then use the artifact instance level interactions extracted to identify interactions between sales orders and invoices. Another solution is proposed but not realized due to the time scope of this thesis. One can merge multiple logs to overcome this limitation of event type level interactions. We can again create an interaction graph for the event logs, and the validness we defined for type level interactions might be reused. Then the logs can be merged using the interaction links.

Conducting case studies has found another limitation of this method which is also mentioned before. Assume there is a trace with an event $e_a$, and the trace interacts with another trace with two events $e_b$ and $e_c$. All three events happened on the same time. Since the ordering of events which has the same timestamps are not deterministic, we could have different traces merged from the two traces, which leads to different interactions even using the same $\text{Miner}(L)$, or different event level interactions identified using the the existence of an event level interactions criterion. If there are multiple similar traces merged differently, the complexity of the resulting model increases unnecessarily.

5.3 Artifact-Centric Model Discovery

In Sections 5.1.2 and 5.2, we have presented the discovery of life cycles and interactions between event types. In this section, we use the life cycles and the interactions discovered to create an artifact-centric process model. Due to the academic background of this thesis, the formal model language proclet system (explained in Section 2.1.3) is used to describe the processes discovered such that formal constructions such as AND-splits, XOR-splits and silent transitions can be used to illustrate and analyze the model behavior. In Section 5.3.1, we show how we create a proclet system. However, since the thesis also has a business context and interests, the model viewed by
users should be easy to understand, which means as simple as possible. Therefore, it is decided to
also visualize a simple representation from a proclet system which is introduced in Section 5.3.2.

## 5.3.1 Proclet System Creation

Let a set \( P_N \) of Petri nets be given, each \( P_N_i \in P_N \) describes the life cycle of artifact \( A_i \). Moreover, a
mapping function \( Q \subseteq (P_N \times P_N) \rightarrow A \times A \) is available, which returns a set \( X \subseteq A \times A \cup A \times A \)
of event type level interactions between two petrinets when given the two petri nets \( P_N_S, P_N_T \in P_N \). In addition, for each petri net \( P_N_i \), we have a function \( AT_i : A_i \rightarrow T_{iv} \) that maps an event
type to the corresponding labeled transition in the Petri net. We present a method which creates a
corresponding proclet system (introduced in Section 2.1.3) by mapping each Petri net to a proclet and
creating the ports for each petrinet and channels between the proclets.

We define the constraint (according to the definition of ports) that every port is connect to
one transition, and each transition is only connect to one input port and one output port. This
constraint is due to the fact that when a transition is connected to two output ports, there is no
explicit expressiveness (identifiable) to distinguish whether a message is sent via both output ports
(AND-splits), or it is only sent via one output port depending on a condition.

First, we create ports for each petri net \( P_N_s \in P_N \) as follows. Given a petri net \( P_N_s \) representing
the life cycle of artifact \( A_s \), we return a union set \( X_s \) of all its event type level interactions towards any
other net \( P_N_i \in P_N \), i.e. \( X_s = \bigcup_{P_N_i \in P_N} Q(P_N_s, P_N_i) \). For each event type \( E_{sOutput} \) of the artifact
\( A_s \), if there is an event type level interaction \( (E_{sOutput}, E_i) \in X_s \) and \( (E_{sOutput}, E_i) \in A_s \times A_i \), we
create an output port for the labeled transition \( AT_s(E_{sOutput}) \) of event type \( E_{sOutput} \) in petri net
\( P_N_s \). For each event types \( E_{sInput} \) of \( P_N_s \), if there is an event type level interaction \( (E_i, E_{sInput}) \in X_s \) and \( (E_i, E_{sInput}) \in A_i \times A_s \), we create an input port for the transition \( AT_s(E_{sInput}) \) of event
type \( E_{sInput} \) in petri net \( P_N_s \).

Now that we have the ports created for the transitions (representing the event types), we connect
the output port of transition \( AT(E_i) \) to the input port of transition \( AT(E_j) \) for each distinct event
type level interaction \( (E_i, E_j) \in X = \bigcup_{P_N_i \in P_N, P_N_j \in P_N} Q(P_N_i, P_N_j) \).

## 5.3.2 Simple representation

Due to the business context of this thesis, interviews have been conducted with advisors of KPMG
to investigate the requirements of clients for the process model. The result of the investigation
has indicated that the proclet system will be too difficult to be understood by the clients or too
time-consuming to be explained to the clients. One of the customers involved in the case study also
indicates that he is less interested in the notions of places, tokens, silent transitions, AND-splits and
OR-splits and finds the sequential relations containing adequate information. Therefore, a simple
representation is used to visualize the proclet system, which is similar to a simple dependency graph.

Given a proclet system, we visualize a simple representation as follows. For each proclet, first,
each place found is removed by connecting each input transition to each output transition of this
place, which results in a graph only consisting of transitions and arcs between transitions. Then,
each silent transition found is removed by directly connecting each of its input transitions with each
of its output transitions. Having all silent transitions removed, we obtain a graph with only labeled
visible transitions and arcs between the visible transitions indicating sequential dependencies. Then,
for each channel found between two ports, we draw an arc from the transition which is connected to
the input port of the channel to the transition which is connected to the output port of the channel.

The simple representation of the OTC example visualizing the proclet system is shown in Fig-
ure 1.7. Each large gray rectangle represents the life-cycle of an artifact visualizing the underlying
proclet (class). An arc within the large gray rectangle represents a sequential dependency between
the event types of the artifact. An arc connecting two transitions crossing over two large rectangles
represents an event type level interaction between the two transitions.

5.4 Artifact-Centric Process Analyses

Business users tend to compare a process model returned by discovery techniques to a process model
they are familiar with in order to detect unusual flows. For example, when a model is discovered
in which an arc is found from the Delivery Created event type to the Sales Order Created event
type, which indicates the deliveries is created before the related sales order, business users shall
immediately spot this unusual arcs and consider this arc as a risk or an outlier. As the meaning of
an arc is ambiguous and depends on the specific discovery technique applied, more information of
the model should be given for users to assess the model.

In this section, we first illustrate several functions to retrieve information when given an event
type level interaction in Section 5.4.1.

5.4.1 Traceability

One of the requirements is that it should be possible to retrieve traces from a given element of the
model. In this section, we first illustrate the information that we can highlight for a chosen event
type level interaction, followed by the information we can highlight for a chosen proclet.

Given an event type level interaction \((E_S, E_T)\), we categorize each combination oriented trace
level interaction \((\sigma_S, \sigma_T) \in I(L_S, L_T)\) into one of the following four types. We assume, the trace
\(\sigma_S\) contains at least one event with event type \(E_S\) and the trace \(\sigma_T\) contains at least one event with
event type \(E_T\):

1. parent trace \(\sigma_S\) is restrictively equal to child trace \(\sigma_T\) on this interaction: for each \(e_s \in F_{E_S}(\sigma_S)\), for each \(e_t \in F_{E_T}(\sigma_T)\), with timestamp \(T(e_s) = T(e_t)\).

2. parent trace \(\sigma_S\) is restrictively eventually before child trace \(\sigma_T\) on this interaction: for each
   \(e_s \in F_{E_S}(\sigma_S)\), for each \(e_t \in F_{E_T}(\sigma_T)\), with timestamp \(T(e_s) \leq T(e_t)\). And there is \(e_s \in F_{E_S}(\sigma_S)\),
   \(e_t \in F_{E_T}(\sigma_T)\), with timestamp \(T(e_s) < T(e_t)\).

3. parent trace \(\sigma_S\) is restrictively eventually after child trace \(\sigma_T\) on this interaction: for each
   \(e_s \in F_{E_S}(\sigma_S)\), for each \(e_t \in F_{E_T}(\sigma_T)\), with timestamp \(T(e_s) \geq T(e_t)\), and there is \(e_s \in F_{E_S}(\sigma_S)\),
   \(e_t \in F_{E_T}(\sigma_T)\), with timestamp \(T(e_s) > T(e_t)\).

4. parent trace \(\sigma_S\) is parallel to child trace \(\sigma_T\) on this interaction: there is \(e_{s1}, e_{s2} \in F_{E_S}(\sigma_S)\),
   \(e_{t1}, e_{t2} \in F_{E_T}(\sigma_T)\), with timestamp \(T(e_{s1}) < T(e_{t1})\), and \(T(e_{t2}) < T(e_{s2})\).

Similarly, it is also possible to categorize a parent oriented trace level interaction \((\sigma_S, \{\sigma_{T1} \cdots \sigma_{T_n}\}) \in I_P(L_S, L_T)\) into the four categories, given an event type level interaction \((E_S, E_T)\). For example,
parent trace $\sigma_S$ is restrictively equal to child trace(s) $\{\sigma_{T1}, \ldots, \sigma_{Tn}\}$ on this interaction: for each $e_s \in F_{ES}(\sigma_S)$ if and only if for each $e_t \in \bigcup_{1 \leq i \leq n} F_{ET}(\sigma_{Ti})$, with timestamp $T(e_s) = T(e_t)$.

In addition, if the merging method is used, we will also be able to give information about the event level interactions found in the merged log using the event level interactions defined in Section 5.2.2. Since the merging method is combination-oriented, the event level interactions are also combination oriented.

When users selecting an event type level interaction (arc), we can retrieve the parent traces which restrictively before, equal to, after of parallel to the child traces of the interaction $(E_S, E_T)$ based on the orientation of interactions to allow users further interpreting, reasoning, and assessing this interactions self.

When selecting a proclet, we illustrate the artifact type level interactions of the artifact represented by this proclet, where the artifact is the parent artifact thus containing the interactions in the log. In addition, we also categorize the trace level interactions (i.e. the artifact instance level interactions of the artifact type level interaction) into four options. (1) the traces of this artifact that have an interaction with the traces which are found in the child artifact log. (2) the traces of this artifact having interactions with the case of the child artifact but not founded in the child log. (3) the traces of this artifact that have no trace level interactions. (4) the traces of the child artifact that have no interactions with the parent traces (i.e. the child traces do not appear in the interaction attribute of the parent traces). When verifying these four types, it was noticed that due to the single parent side trace level interactions extraction, there might be false positives in the fourth option, which means the child trace had a reference, however, the referred parent trace is either not extracted or is not in the data source download. This issue is solvable by extracting the interaction on both sides.

We conclude this chapter by giving an overview of the achievements. Figure 5.2 shows a simple representation of the artifact centric model of the OTC example. In Section 5.1.2, we have shown how to discovery life cycles of artifact by using an existing discovery miner. In Section 5.2, we have presented two different method to discover event type level interactions: one (described in Section 5.2.1) merges trace level interrelated event logs and applies an existing discovery miner to identify the direct successors that are originating from different artifacts as event type level interactions. The second method (described in Section 5.2.2) uses one of the predefined criteria to identify event type level interactions. By using the second method, we have also shown how to identify the set of all possible event type level interactions based on direct succession of events which are found in the merged log and belong to different artifacts (i.e. event level interaction). We used this set to discover unusual flows that are found in this set but not discovered by applying a miner on the merged log (e.g. shown by the red arcs in Figure 5.2). Finally, in Section 5.4, we have shown how to create a proclet system and visualize the proclet system with a simple representation for process analyses, followed by how we highlight the information for analysts to retrieve cases and assess the model.
Figure 5.2: An artifact centric model of the OTC example

*the red arcs represent unusual flows*
Chapter 6

Implementation

In Chapters 3-5, we have shown our approach that addresses (I) the extraction of event logs (with interactions) and (II) automatically mining a proclets system, shown in Figure 1.9. For each part, a prototype is built to implement and test the approach. In this chapter, we shall briefly explain the architecture of the two implementations and the important design decisions. Moreover, screen-shots are given to illustrate the execution of the approach. Since the first part of the approach can reuse the functions of the previous implemented XTract application, it is decided to reuse the implementation and extend the data model and structures to achieve the requirements. The new XTract approach is called XTract2, which is discussed in Section 6.1. The second part is considered to be a mining algorithm. ProM has provided a framework for handling the import of event logs and visualization. Therefore, we decide to implement the mining algorithm as a plug-in for ProM in Java, we called the InteractionMiner.

6.1 XTract2

We have realized our approach regarding the three phases, (A) artifact type identification, (B) type level interaction identification and (C) Log extraction, of (I) the extraction problem (discussed in Chapters 3 and 4) in the XTract2 tool. In this section, we first give an architecture overview of the XTract2 tool and the functionality of each component. Then, examples are given to illustrate the flow of calls. Finally, we show some screen-shots of the tool.

The architecture of the XTract2 tool, illustrated in Figure 6.1, follows in general the model-view-controller architectural pattern. We shall briefly describe the function of each component by going through a flow of calls. The GUI component consists of an UI package which handles all human-machine interactions between users and the XTract2 application. For example, users can specify the internal cache database which should be used, select the source database, can import the schema and change the artifacts and interactions. The GUI component handles the user inputs by calling the packages in the Controller component, i.e. Algorithms, ArtifactController, Data.dataImport and Data.datapersistence. The Algorithms package of the XTract application is unchanged and reused for calculations such as automatic schema identification, artifact (without interactions) creation, and column domain identifications. The ArtifactController handles various changes on the artifacts and their type level interactions. The Data.dataImport package uses the
CHAPTER 6. IMPLEMENTATION

JDBC connector to processes various types of source data, import the data and relational data structure and pass through the Data.persistence package. The Data.persistence package also uses the JDBC connector to handle data insertion, extraction and deletion in the cache database. And both GUI and Controller components uses the data models in the Models component to transfer data within the application.

For example, users specified the data source and the internal cache databases via the UI package, the UI package calls the Data.dataImport package to import the tables, columns and schemas from the source databases and calls the Data.persistence package to create a schema and insert data into the cache database. By calling the functions in the Algorithm package via UI, users can identify artifact schemas and create artifacts. When users modify the artifacts, the UI package calls the ArtifactController to temporarily change the structure of the artifact. Both objects can be stored persistently in the cache database via the Data.persistence package. To recall the artifacts identified, the Data.persistence package extracts data from cache database and uses the Models package to create models which can be demonstrated to users by GUI. When users indicate to extract the artifact via the UI, the UI calls the functions in the Algorithms package, which create a mapping model and used this mapping to create a cache for the OpenXES to write the event log.

![Figure 6.1: Architecture of XTract2](image1)

![Figure 6.2: Architecture of InteractionMiner](image2)

![Figure 6.3: XTract2 - (A1) Import Data schema](image3)

We will illustrate the XTract2 tool based on the three phases of the extraction approach explained in Section 1.4 by emphasizing on the new functionality. The same numbering is used. As the
import data schema’ button in Figure 6.3 shows, we have added the functionality to allow import data schema (A1). The ‘Artifact schema identification by splitting tables’ button calls the new implemented the ComputeArtifactSchemas algorithm to identify artifact schemas. Figure 6.4 shows the interface where users can perform the following tasks:

- add and remove tables of artifact schemas;
- construct artifacts with conditions;
- construct event types with conditions;
- add and remove event types, attributes and interactions;
- calculate type level interactions;
- extract event logs;

![Figure 6.4: XTract2 - (A2) Artifact schema identification and (A3) Artifact identification](image)

The interaction graph is shown in a separate frame illustrated by Figure 6.5. Users can select an interaction in Figure 6.4 to highlight the links of this interactions with red color in the interface shown in Figure 6.5. By clicking the ‘Create Log’ button, the XTract2 tool automatically creates the log mapping (with type level interactions) and an event log for the selected artifact. Figure 6.6 shows a part of the mapping and the event log of the artifact Sales Order.
6.2 InteractionMiner

In this section, an overview of the architecture of the InteractionMiner plugin is first shown. Then, we demonstrate some screen shots of the plugin to illustrate some functionalities.

The architecture of the InteractionMiner application, illustrated in Figure 6.2, follows a general ProM plug-in pattern, which has a MinerPlugin package, a Visualizer package, a LifecycleMiners and a MergerMiners package. Given a set of XES event logs (with artifact instance level interactions) as input of the MinerPlugin package, the MinerPlugin package calls the specified lifecycle miner from the LifecycleMiners package to construct a proclet for each log, then the MinerPlugin merges the logs, exports the merged log as an object to the ProM framework and creates a connection to be able to retrieve the merged log. The MinerPlugin calls the specified MergedLogMiner in the MergedLogMiners to apply it on the merged log which returns event type level interactions identified. The MinerPlugin then returns a proclets system (called artifactModel) to the ProM framework, which will automatically call the corresponding visualizer to visualize the proclets system. Users can interact with the visualizer package to retrieve information or select different interactions.

We illustrate our mining approach by going through the screen shots of the InteractionMiner plugin. First, users can load a set of event logs into ProM and select the logs for applying the InteractionMiner plugin, as shown in Figure 6.7. Then, users can specify the life cycle mining algorithm and the merged log mining algorithm in the dialog shown in Figure 6.8. If the selected algorithm has other settings (e.g. heuristic miner has a dialog to set parameters), these will also be shown. After the miner is finished, we obtain an artifact centric model. Users can select the proclet view shown in Figure 6.9 or the simplified representation shown in Figure 6.10 in the visualizer. Furthermore, the right hand side of the visualizer allows users to select a subset of the transitions, to select a criterion for event type level interaction identification, to obtain more information of an element selected in the model, and to view different flows between two artifacts that have interaction with each other.
Figure 6.7: Selecting event logs for InteractionMiner

Figure 6.8: The dialog to select the life-cycle miner and the merged log miner

Figure 6.9: The proclet system view of the artifact model in the visualizer

Figure 6.10: The simple representation view in the visualizer
Chapter 7

Case Studies

We have defined our approach and shown the tools within previous chapters. To verify and validate our approach and to use the process models obtained to perform analyses, two case studies were conducted. In this chapter, for each case study, we shall (1) introduce the process and the data source, (2) explain the execution of our approach including decisions made, and (3) discuss observations and results we have obtained.

The first case study was performed in the OTC process of SAP. In this case study, we emphasize on the execution of our approach (such as decisions about the artifact selection and interaction selection) since no customer was involved. The second case study was conducted in the PA process of Oracle. Since customers were involved, we emphasize the discussion of the result of analyses. For both case studies, we are guided by the advisors of KPMG, who are the experts in ERP systems and data analytic and have rich experience of conducting advisory projects for clients.

7.1 Case I - SAP Order To Cash Process

The first case study was performed for the Order to Cash (OTC) process supported by SAP systems. The data is provided by KPMG. Since no specific customer was involved, the requirements of the case study for the OTC process of SAP were provided by the advisors of KPMG, who are the experts in SAP systems and have rich experience of conducting advisory projects for clients. Therefore, the result obtained is also validated and discussed with the advisors. In this section, we first give a short introduction to the OTC process and the relational data structure that is used by SAP. Then, we emphasize the approach in which we obtain event logs and artifact models of the OTC process. Finally, we discuss some observations about the models obtained.

7.1.1 SAP OTC process and data structure

A simple OTC process in the system starts with creating a sales order. After the order is delivered, a delivery document is created. Then, an invoice document is created in the system, sent to customer and posted in the account receivable (table). After receiving the payment, this simple process ends. However, there are many complex variations of this process. For example, the orders could be linked to a contract document, or return orders could be placed and return deliveries are made. Credit
memo requests might be received from customers when the goods are incomplete or damaged. Invoices might also be canceled.

The data structure used to support the OTC process allows flexibility to deal with the aforementioned variance, but it is also very complex. The relational data structure of SAP regarding the relevant tables of this case study is shown in Figure 7.1. The VBAK table stores various sales documents such as sales orders, inquiries, contracts, return orders, credit memo requests and debit memo requests. As these documents share similar structure, they are put into one table, and the value in the column vbtyp indicates the type of a document. Similarly, the delivery documents such as deliveries, return deliveries and delivery shipping notifications are stored in the LIKP table, and the invoice documents such as invoices, invoice cancellations, inter-company invoices, and credit memos are stored in the VBRK table. Each document has lines stored separately. For example, a sales order of a web-shop might have included different products, where each line represents a product (or material) with the ordered quantities. The lines of the sales documents (in VBAK), the delivery documents (in LIKP), and the invoice documents (in VBRK) are stored in the tables VBAP, LIPS, and VBRK, respectively. Documents are linked through lines. For example, a delivery (document) could combine various lines from various sales orders which are in the stock. Invoice documents are then placed in the account receivable tables BKPF and BSID as invoices with open payments. When payments are received, the open payment in the BSID table is deleted and added to the closed payment table BSAD. Each document and each line could have separate changes performed and stored in the change tables CDHDR and CDPOS. For example, a billing block release is a change of sales documents, whereas price changes are related to the sales lines.

![Figure 7.1: SAP OTC process - relational structure](image)

7.1.2 SAP OTC - Extraction and Discovery

In this section, we emphasize the steps taken and decisions made to obtain an artifact centric model for the SAP - OTC process.
First, we imported 11 tables that constitute the basic data of the OTC process. Figure 7.2 shows the name of the tables, the constraint for selective import, the number of records used, and the number of columns. The time scope of the creation of documents is set between ‘01-09-2012’ and ‘31-10-2013’. In contrast to Oracle, SAP used abbreviated name and codes as table names, column names and values, which are difficult to understand. To have readable names of the change event types identified from the CDPOS table, we had to manually create a column EventName in the CDPOS table that translates the value in the fname to an understandable name which indicates the type of a change. Then, we imported the primary keys and the foreign keys (respectively, shown in Figures C.1 and C.2 in Appendix C.1) of these tables and identified the data schema. A timestamp in SAP is normally separated into a date column and a time column. Since some time columns in the OTC process (e.g. the time column of invoice documents - VBRK.ERZET) were not downloaded by KPMG, we only consider ‘date-stamps’.

We identified eight artifact schemas. Based on various trials performed before this case study, we concluded that it is much better and reasonable to identify the documents and the lines as separate artifact schemas and artifacts. We observe that each line in the document has its own life cycle, different from the document itself. For example, a sales document might be created and delivered, but some lines in this document might be rejected and not delivered, others might have a price change. Moreover, by including the events of lines to documents, the complexity of an artifact increased enormously, which lead to an unreadable model and worsens the data convergence and divergence problem. For example, the sales order artifact including the lines could have hundred different types of changes as demonstrated in Figure 7.3. Furthermore, the advisors were only interested in documents and claimed that information of lines might be too much detail for client. In addition, considering the documents and the lines as separate artifacts decreases the size of artifacts, which improves the performance of the extraction of log and the merge of logs. For these reasons, we split the documents and the lines table (shown in Figure 7.1) into different artifact schemas (which is also done by the algorithm). We only include the changes of sales documents and invoice documents, which are the interesting changes indicated by the advisors. Therefore, we obtain the following eight artifact schemas shown in Figure 7.4.
### CHAPTER 7. CASE STUDIES

Figure 7.3: SAP OTC process - sales order lifecycle including the changes of lines

<table>
<thead>
<tr>
<th>Artifact Schema</th>
<th>Maintable</th>
<th>Reflect</th>
<th>Artifact function</th>
<th>Expected</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSKF</td>
<td>SKIF</td>
<td>ProductIDI</td>
<td>maintable_bsisp = 'SKIF'</td>
<td>Yes</td>
</tr>
<tr>
<td>BRAD</td>
<td>BRAD</td>
<td>ProjectID</td>
<td>maintable_bsisp = 'BRAD' or maintable_bsisp = 'BI'</td>
<td>Yes</td>
</tr>
<tr>
<td>LKIP</td>
<td>LKIP</td>
<td>Delivery</td>
<td>maintable_brlln = 'L' or maintable_brlln = 'I'</td>
<td>Yes</td>
</tr>
<tr>
<td>LIPS</td>
<td>LIPS</td>
<td>DeliveryL</td>
<td>inner join tableBPRV y1 on maintable = y1 and y1.brlln = 'L'</td>
<td>Yes</td>
</tr>
<tr>
<td>LIPS</td>
<td>LIPS</td>
<td>DeliveryL</td>
<td>inner join tableBPRV y2 on maintable = y2 and y2.brlln = 'I'</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Figure 7.4: SAP OTC process - artifacts
We identify one or multiple artifacts for each artifact schema as illustrated by the third column in Figure 7.4. For the BKPF artifact schema which describes payment documents, only one artifact is identified. The advisors indicates that the value in the bschl column of the closed payment table (BSAD) refers to the type of payment. When the value in the bschl column is 05 or 15, it indicates that the record is a real payment received. Therefore, a separate artifact type is created. For the sales documents (VBAK) and lines (VBAP), delivery documents (LIKP) and lines (LIPS), and invoice documents (VBRK) and lines (VBRP), there is a column vbtyp in the document tables that indicate the type of documents. Therefore, we created the artifacts of documents and lines based on the values found in this column. In total, we have identified 35 artifacts.

To identify and select the artifact type level interactions, several decisions were made based on the interviews with advisors. First, only the documents were interesting for the advisors, thus line artifacts and their interactions were omitted. This requirement and the OTC table structure shown in Figure 7.1 indicates that a weak join has to be included. For example, a sales document must first join the sales lines, then join the delivery lines, to join the delivery document, in which the last join is an over-approximation. Second, there is a type of foreign keys in the sales, delivery and invoice documents that indicate the previous document(s) of a record, i.e. the vgbel and vgpos columns. This previous document relation is very interesting since the relation indicates the transformation (or causality) between different type of documents. Therefore, if this foreign key existed, we select the interactions that are based on this reference. Furthermore, since the main goal is to identify outliers, any interaction with at least one record is selected. Thus, we have set the least number of record \( r = 1 \), the number of strong joins \( k = 2 \) and the number of weak joins \( m = 1 \). Figure 7.5 shows a screen shot of the interaction graph that is constructed. The type level interactions that are selected and extracted are shown in Figure 7.6.

For each artifact shown in Figure 7.6, we obtain an event log with trace level interaction. In total, 18 event logs are extracted and imported into ProM. Using the heuristic miner as the life-cycle miner and the merged log miner, we obtain the artifact-centric system shown in Figure 7.7. Each life cycle only contains one event type to first have a simple helicopter view. Thus, for all artifacts except artifact 'payment05or15', we first only considered extracting the Creation event type. For the artifact 'payment05or15', we only considered the clearing date (i.e. the column augdt).
decision is also based on various trials. Since business users would first like to have a helicopter view of the process, it is better to keep the overview as simple as possible. Furthermore, the Creation event type was set by the system, thus the timestamps of this event type is reliable, and users can not manually change it. Furthermore, it is easy to add event types (e.g. changes), extract a more comprehensive artifact and replace the simple artifact with the extended artifact. We will illustrate one example of this replacing approach. After extracting the simple Order artifact with only an event type Created, we add the change event types shown in Figure 7.8 found in the change tables CDHDR and CDPOS to the artifact using our approach (described in Section 3.3). We use the manually created column EventName as the event type name to split the event type and to have readable names of the change event types. Without changing any interactions, the extended Orders artifact can be extracted again.

![Figure 7.6: SAP OTC process - artifacts with type level interactions extracted](image)

![Figure 7.7: SAP OTC process - artifact-centric model with simple representation](image)
7.1.3 SAP OTC - Process Analyses and Discussion

In this section, we discuss three observations: identifying unusual flows, extending and replacing artifacts and the complexity of interactions.

One of the analyses is to identify unusual flows. In the "Interaction Filter" tabbed panel on the right hand side of the InteractionMiner visualizer, we can select the ‘show all interactions’ option and obtain all event type level interactions using the existence of an event level interactions criterion (described in Section 5.2.2). Moreover, the unusual event type level interactions which are obtained by using this criterion but not discovered by the miner (also described in Section 5.2.2) are colored red. The model shown in Figure 7.9 demonstrate only one unusual flow (indicated by the red arc): there were payments received before the invoices were created in the account receivable.

Verifying the cases retrieved via this unexpected flow in the database with the query in Listing 7.1 has shown that the Payment Received date is indeed earlier than the Posted In AR event. Advisors
have also validated that this flow is strange and can not be explained other than the date were manually changed. Further investigation conducted by advisors has proved that the cases was indeed manually changed by someone using the transaction code FB05, which is not a usual action (which has real financial risks). In other words, we have successfully and exploratively discovered a true-positive unusual action.

Listing 7.1: A query verifying payments received date earlier than invoice placed in AR

```
SELECT a.BELNR, a.CPUDT, a.AWKEY, b.BELNR, b.BSCHL, b.AUGBL, b.CPUDT, b.AUGDT, c.BELNR, c.cpudt, c.AUGDT, c.BSCHL
FROM bkpf a
INNER JOIN BSAD b ON b.BELNR = a.BELNR and b.gjahr = a.gjahr and b.bukrs = a.bukrs and b.mandt = a.mandt
INNER JOIN BSAD c ON c.belnr = b.AUGBL and b.gjahr = c.gjahr and b.bukrs = c.bukrs and b.mandt = c.mandt
WHERE a.CPUDT > c.AUGDT and (b.bschl = '01' or b.bschl = '11') and (c.BSCHL = '05' or c.BSCHL = '15') and a.awtyp = 'vbrk' and a.belnr = 'x' -- x = caseID
```

The second observation is that, after creating the artifact centric model shown in Figure 7.7 which is simple and easy to analyze, we can now replace simple artifacts with extended artifacts to show more detail of the process. For example, we have created a more comprehensive artifact Order by adding all change event types found in the change tables (described in Section 7.1.2). Since the trace identifiers have not changed, and interactions have not changed, there is no need to reconstruct other event logs. We replace the simple event log of Order with the comprehensive event log of Order, and obtain an artifact centric model shown in Figure 7.10.

![Figure 7.10: SAP OTC process - artifact-centric model with the discovered interactions](image)

Note that we can now identify some difference in event type level interactions between the sales order artifact and other artifacts.

To show this difference more clearly, we obtain a more simplified model shown in Figure 7.11 by using the **max number of event level interactions** criterion, that is, we only highlight type level interactions where the number of event level interactions is maximal between two artifact types.
The red circles indicate an interesting difference. For example, most artifacts, e.g. delivery, credit memo request, debit memo request are generally created directly after the creation of sales order, whereas the invoices (that are directly related to a sales order via vgel and vgapos) are created after the Release Date change event type of this sales order. Similar for the invoice cancellation artifact, of which the creation generally takes place after the Next Date of the sales order changes.

Only to illustrate the complexity, we have included the model with all direct interactions shown in Figure 7.12. The red arcs are the unusual event type level interactions which are obtained by using the existence of an event level interactions criterion but not discovered by the applying the miner on the merged log (described in Section 5.2.2). As this model is also almost impossible to analyze, further simplification techniques have to be developed. For example, visualization techniques might help improve the understandability of the model such as the edge bundling techniques proposed by D. Holten and J. van Wijk [15].
7.2 Case II - Oracle Project Administration Process

The second case study is performed for the project administration (PA) process supported by the Oracle information system of an educational organization. The data is downloaded from the educational organization provided by KPMG. This case study is done based on the request of the client, i.e. the educational organization. Therefore, feedback received from the client is also discussed. First, a brief introduction to the project administration process is given, together with an overview of the tables that were downloaded and available for analysis.

7.2.1 Oracle PA Process and Data source

An educational organization has thousands of projects running, e.g. different research projects. The project administration process supported by Oracle starts with creating projects in the system. At the moment of creating a project in the system, it is usually definitive that the project will be executed. It is possible that the project has already started. After the project is created in the system, one can specify relevant information of the project such as its starting date. During the execution of the project, tasks are created for the project to declare different expenditures related to a task, e.g. personnel, materials. Interviews with advisors have indicated that, for assessing financial risks, it is important that the ending date of expenditures is before the the complete date of tasks. Moreover, all tasks should be completed before the completion date of a project. When a project is completed, it means that the main activities, such as the research itself, are finished. When the administration work is completed, such as financial checks, the project is closed.

For this case study, 18 tables were downloaded, and 7 tables were used in the actual analysis as shown in Figure 7.13. Due to the time constraint, we have limited our data scope between 01-06-2012 and 31-12-2012. The number of the records of each table used in the process analysis is also shown in the fourth column of Figure 7.13.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Row Count downloaded</th>
<th>Used</th>
<th>Row Count used</th>
<th>Column Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>FND_USER</td>
<td>905</td>
<td></td>
<td>905</td>
<td>27</td>
</tr>
<tr>
<td>HR_ALL_ORGANIZATION_UNITS</td>
<td>1053</td>
<td></td>
<td>1053</td>
<td>43</td>
</tr>
<tr>
<td>MTL_SYSTEM_ITEMS_B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OF_ORDER_HEADERS_ALL</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PA_COST_DISTRIBUTION_LINES_ALL</td>
<td>95943</td>
<td>x</td>
<td>5543</td>
<td>15</td>
</tr>
<tr>
<td>PA_EXPENDITURE_COMMENTS</td>
<td>55149</td>
<td></td>
<td>30511</td>
<td>7</td>
</tr>
<tr>
<td>PA_EXPENDITURE_ITEMS_ALL</td>
<td>96978</td>
<td>x</td>
<td>5620</td>
<td>32</td>
</tr>
<tr>
<td>PA_EXPENDITURE_TYPES</td>
<td>94</td>
<td>x</td>
<td>94</td>
<td>10</td>
</tr>
<tr>
<td>PA_EXPENDITURES_ALL</td>
<td>682590</td>
<td>x</td>
<td>3100</td>
<td>24</td>
</tr>
<tr>
<td>PA_PROJECT_CUSTOMERS_V</td>
<td>16238</td>
<td></td>
<td>16238</td>
<td>17</td>
</tr>
<tr>
<td>PA_PROJECT_STATUSES</td>
<td>80</td>
<td>x</td>
<td>80</td>
<td>23</td>
</tr>
<tr>
<td>PA_PROJECTS_ALL</td>
<td>5364</td>
<td>x</td>
<td>1132</td>
<td>29</td>
</tr>
<tr>
<td>PA_TASKS</td>
<td>2416</td>
<td>x</td>
<td>1236</td>
<td>31</td>
</tr>
<tr>
<td>PA_TRANSACTION_SOURCES</td>
<td>48</td>
<td></td>
<td>48</td>
<td>4</td>
</tr>
<tr>
<td>PAY_COST_ALLOCATION_KEYFLEX</td>
<td>1694</td>
<td></td>
<td>1694</td>
<td>11</td>
</tr>
<tr>
<td>PO_HEADERS_ALL</td>
<td>5186</td>
<td></td>
<td>5186</td>
<td>139</td>
</tr>
<tr>
<td>PO_LINE_LOCATIONS_ALL</td>
<td>7700</td>
<td></td>
<td>7700</td>
<td>148</td>
</tr>
<tr>
<td>PO_LINES_ALL</td>
<td>7700</td>
<td></td>
<td>7700</td>
<td>135</td>
</tr>
</tbody>
</table>

Figure 7.13: Oracle PA process table record counts
We briefly describe the eight tables used. Each project corresponds to a record in the PA_PROJECTS_ALL table. The timestamp of the creation of a project is stored in the CREATION_DATE column. The starting date, completion date, and closed date of a project are stored in the START_DATE, COMPLETION_DATE and CLOSED_DATE columns, respectively. Each task is stored as a record in the PA_TASKS table, which has a foreign key to the PA_PROJECTS_ALL table. Expenditure documents are stored in the PA_EXPENDITURES_ALL table, and expenditure items of an expenditure document are stored in the PA_EXPENDITURE_ITEMS_ALL table. The expenditures are linked to the tasks via the PA_EXPENDITURE_ITEMS_ALL table directly. The PA_COST_DISTRIBUTION_LINES_ALL table also provides a link between the tasks and the expenditure items. Each expenditure item has an expenditure type which can be categorized into eight different expenditure category, e.g. ‘Third-party services’, ‘Equipment, Inventory and Software’, ‘Housing Expenses’. Information of expenditure types and categories are stored in the PA_EXPENDITURE_TYPES. Each project has a current status (e.g. rejected, approved, closed), and information of the status are stored in the PA_PROJECT_STATUSES table.

A document of the tables was provided by advisors of KPMG, which includes information of the primary keys and foreign keys of the tables, helping us to obtain the data schema. Other tables contain more informative date and are less relevant for the core PA process.

7.2.2 Oracle PA - Extraction and Discovery

We first have identified 6 artifacts. Since the information stored in the PA_PROJECT_STATUSES table and the PA_EXPENDITURE_TYPES table are only informative and not related to a specific step of the process, we only considered the other six tables. Each of the six tables is then considered as an artifact schema and is mapped to one artifact, respectively. The six artifacts are shown in
CHAPTER 7. CASE STUDIES

Figure 7.15. Applying the approach described in Section 4.2.1, seven direct type level interactions are found between the artifacts which are shown in Figure 7.14.

<table>
<thead>
<tr>
<th>Artifacts</th>
<th>Maintable</th>
<th>Extracted</th>
<th>Interaction From</th>
<th>To</th>
<th>via</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>PA_PROJECTS_ALL</td>
<td>x</td>
<td>Project</td>
<td>Tasks</td>
<td></td>
</tr>
<tr>
<td>ExpAll</td>
<td>PA_EXPENDITURES_ALL</td>
<td>x</td>
<td>Tasks</td>
<td>ExpAll</td>
<td>ExpItem</td>
</tr>
<tr>
<td>Tasks</td>
<td>PA_TASKS</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CostDistr</td>
<td>PA_COST_DISTRIBUTION_LINES_ALL</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExpItem</td>
<td>PA_EXPENDITURE_ITEMS_ALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExpTypes</td>
<td>PA_EXPENDITURE_TYPES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.15: The artifacts created for Oracle PA process

Figure 7.16: Three event logs extracted

For the log extraction step, since the project administration (PA) process mainly consists of the projects, tasks and expenditures, we extract these three artifacts and type level interactions between them, demonstrated by Figure 7.15. The three event logs are imported into ProM. Figure 7.16 shows the number of occurrences of traces and events of the three logs. We observe that each number of traces (or cases) of an event log confirms the number of the row count of the corresponding main table.
For the artifact-centric process discovery, the heuristic miner is used as the life cycle miner and as the merged log miner. We obtain the following proclet model in simple representation shown in Figure 7.17. Using the **max number of event level interactions** criterion defined in Section 5.2.2 to identify the event type level interactions, we obtain a simpler model shown in Figure 7.18 which can be more easily explained to business users.

Figure 7.17: A proclet system discovered using merging logs method

Figure 7.18: A proclet system discovered by using the max number of event level interactions criterion
7.2.3 Process Analyses Result and Discussion

As a goal of the thesis is to perform process analysis using the artifact centric approach, the results obtained are discussed and validated with advisors. We have found a set of unusual flows using the artifact-centric model. We were able to retrieve the cases for each unusual flow we found and verified them in the original database. Then, an interview with the client was conducted, during which the clients gave explanations for these unusual flows.

In this section, we first discuss the unusual flows found between the artifacts. Then, we shortly discuss the unusual flows found within the artifact life cycle. Finally, additional questions asked by the client and feedback given by the client regarding the model are presented. We discover the following unusual flows between the artifacts, noted using a unique numbering that corresponds with the numbering shown in models:

- First, projects must be created in the system before tasks, and tasks should be created before expenditures. Therefore, we select the creation event types of three artifacts and use the **existing precedence** criterion (explained in Section 5.2.2) to see if any unusual flow can be observed regarding the creation time. As the model obtained in Figure 7.19 shows, no tasks were created before the creation of the related project (1). However, there are expenditures created before the creation of the related task (2).

![Figure 7.19: A proclet system discovered by using the existing precedence criterion](image)

Both the advisors and the client indicate that this unexpected flow is possible in the following situation: when a project has started, and expenditures have already been made during the initial phase of the project and have to be declared but no specific tasks are created yet. The same explanation is given when we observed that there are expenditures created before tasks started. This result indicates that our approach is able to identify various ordering of event types via the event type level interactions.
Some unusual flows were observed by advisors between the Start Date and Creation Date of projects and tasks. Therefore, we have included the Start Date of projects and tasks (3) and obtained the model shown in Figure 7.20.

![Figure 7.20: A proclet system discovered by using the existing precedence criterion](image)

The client, who is also the business expert, indicated that it is normal to have a project created before or after the project is started. For example, they might have waited with creating a project in the system until the funding of project is received whereas the research project has already started. The same reason holds for the tasks. However, the Start Date of tasks should always be after or equal to the Start Date of project, which is true as the model shows (4). This observation has also led to the next question from the client.

The creation of projects and tasks in the system are administrative work. The client indicates that when creating a project in the system, the specification of all tasks related to this project are known, thus the creation time of tasks should happen shortly after the creation of the project. If not, it may indicate that double administrative work have been done. Since the average time of different event type level interactions is calculated for the shortest time criterion defined in Section 5.2.2, we were able to retrieve this the average time between the creation of project and tasks (5) easily, which is 1.088 day. However, the average time might be an inaccurate indication. Therefore, we asked the client to indicate a maximal threshold. The client indicates the time between the creation of project and the creation of related tasks should be less than or equal to two weeks. We had to verify this property manually in the database. Of the 1236 tasks, we found 1197 tasks which were created less than a day after the project was created; 8 tasks which were created between a day and 14 days after the project was created; and 31 tasks which were created greater than 14 days. We found no task was created before the creation of its project (5). This result show that our approach
has correctly shown the flows. However, more details and data might be necessary to help
users assess a certain interaction. We consider the visualization of the more detailed data of
a certain interaction as future research.

• Both the advisors and the client indicate the closing of project process should follow this
ordering: expenditure ending, task completed, project completed and finally project closed.
As the model in Figure 7.21 shows, we have found one task which is completed before the two
expenditures that are related to this task (6).

![Figure 7.21: A procler system discovered by using the existing precedence criterion](image)

Consulting the Oracle website\(^1\) by advisors, the expenditure could end in the weekend of the
same week that the task is completed. Verifying this constraint in the original database, it
indeed shows that the expenditure is created at Sunday, two days after the task is completed.
Thus, this case is allowed. Our artifact centric approach has identified this unusual flow purely
based on the data. However, to understand that this unusual case is allowed, more domain
knowledge might have to be built into the model.

• One observation which can not be explained by advisors and client is that there are many
projects closed before the related tasks are completed (7). This ordering of events indicates
the risk that expenditures can be booked on the tasks while the project is already closed.
Therefore, the client has asked us to further investigate whether there are expenditures created
after the projects are closed or completed. We replace the event log of projects which has trace
level interactions to tasks with another event log of projects which has trace level interactions
to expenditures and obtained the model shown in Figure 7.22.

\(^1\)http://docs.oracle.com/cd/A60725_05/html/comnls/us/pa/dates06.htm
We found two projects that are created in parallel with the creation of the expenditures (8). The definition of parallel (given in Section 5.4.1) indicates that there are expenditures which are created after the two projects are closed. Verifying in the database, we retrieved five expenditures that were indeed created after the two projects were closed, shown in Figure 7.23. This result again shows that our approach is able to illustrate true positive unusual flows.

Also, unusual flows have been found within an artifact life cycle. During the validation with advisors, we have found some limitations about the information shown within an artifact. One limitation found is that the Creation and Last Update timestamps recorded by Oracle have the time information, whereas the other timestamps such as start, complete and closed only have the date information. This difference has led to unusual ordering of the events, which led to false positive unusual flows. For example, the Creation of a project is at 2012-08-01 10:26:34.000, and the Closed Date of the project is at 2012-08-01 00:00:00.000. This problem is caused by the limitation of the data source. One could build the functionality to categorize cases into two categories, of which the time duration between the two sequential event types is within a day or longer than a day. Note that this limitation is also relevant for the information shown for the interactions. We consider this limitation as future work.

Another problem found is also related to timestamps. To assess the risk, it is important to
distinguish between the situation that an event really happens before another event and the situation that an event happens at the same time as another event. We have added the functionality to distinguish this situation. However, this functionality has led to another problem observed. When a user selects a sequential relation (i.e. arc) between two event types, we only display the cases which contains two events direct followed by each other, and not the cases in which two events is eventually followed by each other:

- For example, we have only found one project which is closed before the project is completed. However, when verifying this observation in the database, we have found more projects which are closed before the projects are completed. This example illustrate what we have discussed, we have not displayed the cases, of which projects are first closed, then other event(s) happened, followed by the Completed event. Assessing the impact of the unusual flow when a project is closed before completed, the false negative cases have to be included, which is considered as future research.

- We have observed one task which is created after the task was started and closed, but no expenditure is declared for this task. Verifying the task in the database, we have found that the Closed Date is more than one day before the creation of the task. We have verified that there are no false negative cases that are not detected due to the same aforementioned reason. However, both advisors and client can not explain this observation with valid reasons. This result again shows that our approach is able to illustrate true positive unusual flows. Since it is only one task and no related expenditure is found in the database, the client indicates that this observation was less relevant.

- A unusual flow, not indicated by the advisors but observed by the client, was that there is a sequential dependency between the Closed Date and the Last Update of project, i.e. there are updates on projects after the projects are closed. The client indicates that no change should be found after projects are closed. This unusual flow might be due to the limitation that the timestamps of closed events only have a date and no time. We verified this on the database. We found that all projects have the last update later than the closed date, but only four projects of which the closed date is more than one day earlier than its last update, shown in Figure 7.24. The result and the verification in the database actually indicates that our approach purely illustrated the data, whereas the quality of the data has an impact on the model discovered. Missing the time element in some timestamps causes unusual flows.

![Figure 7.24: Four projects of which the closed date is more than one day earlier than its last update](image)

We have received the following feedback to refine and extend the model.

- It had been possible to relate purchase orders (which can be found in the Purchase Order (PO) tables) and accounts payable invoices (which can be found in the Accounts Payable (AP) tables) to the expenditures, indicated by advisors. This linkage may have been very
interesting because it indicates the financial impact. However, no AP tables were downloaded and no link was found between the project tables and purchase order tables in the given data source. Moreover, the client indicates that due to the organization characteristic, the creation of invoices varies very much, especially compared to the process of project and tasks. Depending on which project, sometimes, it is allowed to create AP invoices in the beginning. In other cases, for instance, in EU research projects, an invoice might be created in the end of projects by collecting all expenditures. As a result, the assessment of the risks or impact of the execution process of invoices will have to be done per case, which is too much work and too much detail, indicated by the client. Since the client found this extension less interesting, no further analysis was performed.

- Both advisors and client find splitting the expenditures based on the expenditure categories might be very interesting. They also indicate that the personnel cost is very constant during the execution of projects, thus less interesting, but all other categories are much more interesting to analyze. We can easily divide the expenditure items into different artifacts based on the EXPENDITURECATEGORY column by joining the PA_EXPENDITURE_TYPES table. We obtain eight artifacts shown in Figure 7.25.

![Eight artifacts obtained from expenditure items](image)

Figure 7.25: Eight artifacts obtained from expenditure items

The model shown in Figure 7.26 confirms the assumptions that each type of expenditure has different event type level interaction with the project life cycle. For example, the staff expenditures are more created at the beginning of the project (9), whereas the others expenditures are created after the project is definitive (i.e. created in the system) or after the Update Program event type(10) of the project.

In addition, the understandability of the model is discussed. The client indicates that when he received the model for the first time without any further explanation, it was still very hard to understand such a process model. Especially, business users were used to static diagrams such as histograms, or pie charts. But after an hour interactive session (i.e. the interview) together with us have helped client (who is the business expert) to clearly understand the process model, which was afterward easy to analyze. A good argument for this claim is that the client was able to observed the unusual flow from the Closed event type to the Last Update event type of the project which was not indicated by advisors before the interview. The client indicated that the number of cases are also important to help assess the impact of certain unusual flows.

In the beginning of the interview, the client has shown us a diagram in which the budget allocations and the budget realizations of projects are compared. It is interesting to list the possibility of using process models to explain the difference between the allocation and realization as future work.
We conclude this chapter by summarizing the results of the two case studies. For both case studies, we were able to successfully create the desired artifacts and identify the desired type level interactions. We were able to use the event logs which were extracted from these artifacts and interactions to successfully discover an artifact-centric model. Moreover, for both case studies, we were able to use the discovered model to identify true-positive unusual flows validated by the advisors and client. However, there were also some limitations. In some cases, domain knowledge might be required to distinguish the "allowed" flows from the real unusual flows identified. Moreover, the understandability of the model can be further improved. Furthermore, the InteractionMiner tool should visualize more data of a certain flow to help users assess the flow accurately.
Chapter 8

Conclusion

Within this master thesis, we have analyzed, using an artifact-centric approach, the extraction of event logs with trace level interactions from relational data sources and the mining of artifact-centric models from these event logs. We recall the goal of this master thesis and the research outline shown in Figure 1.9:

*Given a relational data source, we would like to support business analysts to (semi-) automatically discover an artifact-centric model, which describes the process of several collaborative processes (or life cycles of artifacts), each of which has its own life cycle, and interactions with each other, and to be able to use this discovered model to perform process analyses.*

Before presenting our approach, we have investigated each element in the research scope: event logs, process models, relational data sources, and more importantly, the traditional approach of log extraction and the original XTract approach of artifact-centric log extraction. Emphasizing the limitations of traditional approaches, we have shown that the data convergence and divergence problem was unsolved, and that these approaches were less suitable for data- (or document-) centric systems by considering the lifecycle of various documents as one process definition. We have also shown the limitations of the original XTract approach such as run-time issues and inability to identify the desired artifacts. Furthermore, no interactions were identified, thus no artifact-centric process model is discovered.

Our first contribution is the extension of the data structure of the original XTract approach and functionality added to allow users to identify the desired artifacts from a given data source. Users are now able to (1) import primary keys and (complex) foreign keys, (2) change artifact schemas, (3) identify various artifacts that share the same artifact schema, and (4) change artifacts by adding or removing event types and attributes.

The major contribution we made is categorizing the interactions, identifying type level interactions on the database level and extracting instance level interactions from event logs. We have divided the interactions into four types: type level interactions, instance (or trace) level interactions, event type level interactions and event level interactions. We have formally defined valid artifact type level interactions and developed a method to identify these interactions. Furthermore, we have extended the original XTract implementation to support (1) identifying type level interactions,
CHAPTER 8. CONCLUSION

(2) allowing users to select the desired interactions for each artifact, and (3) extract an event log with instance level interactions for an artifact. Event logs with instance level interactions can allow more general process mining techniques to be applied and developed, independent of the discovery techniques we proposed.

The last achievement is the artifact-centric process discovery algorithm we proposed and implemented as the InteractionMiner plug-in in ProM. We were able to automatically discover an artifact-centric model in the proclet language by: (1) identifying the life cycle of each artifact using existing discovery algorithms and (2) identifying the event type level interactions through merging the interacting logs and applying existing discovery algorithms. Furthermore, the InteractionMiner visualizer allows users to select criteria and the desired transitions to further explore the artifact-centric model. Also detailed information are provided for users to assess and verify the unusual behavioral patterns identified.

To verify and validate our approach and the artifact-centric models we obtained, two case studies have been performed on two different process of two different ERP systems. The first case study, performed within the OTC process of SAP, has shown that our approach is able to deal with complex data structure and identify complex artifacts and type level interactions. The second case study, performed within the PA process of Oracle, has shown that the customers were able to understand the model and discover a unusual behavioral pattern by themselves. Both case studies have shown that the approach was able to successfully discover the desired artifacts and the artifact-centric process model.

We believe that we have developed an artifact-centric approach. Our prototypes support business analysts to (semi-)automatically discover an artifact-centric model from a given relational data source. We also believe that we have shown that the artifact-centric model discovered is more intuitive than the old approaches for business users and can be used to perform process analyses.

8.1 Limitations and Future Work

In this section, we discuss the limitations and future work with respect to the following four aspects: (1) artifact identification, (2) interaction discovery, (3) process analyses and visualization, and (4) functionality of the tools.

Artifact Identification

For the (A) artifact type identification phase of the extraction problem, we list the following limitations and possible extensions. First, we are still unable to automatically identify the “perfect” artifact (schema). For example, it is more logical to include the change tables (CDHDR and CDPOS) of SAP in artifacts, instead of considering them as a separate artifact. However, this solution causes the data divergence problem. We allow users to change the artifact (schema) identified by our approach but this also increased the burden on users. Domain knowledge is really necessary to create the desired artifacts for conducting process analyses. Therefore, more support should be provided for users.

A possible solution could be using pattern recognition/matching techniques, which build a pattern repository of (im)possible structures of an artifact and use these structures to automatically
identify (possible) artifacts. Another solution could be implementing more heuristic rules. For example, the columns which has a certain amount of values (e.g. \( \geq 1 \land \leq 10 \)) can be used for splitting the artifacts or the event types.

We notice that both SAP and Oracle systems have a document (header) and document line structure. For example, the sales documents and the sales lines. The advisors are typically interested in the document (header) level. It might be possible to introduce an approach to level the (main) tables (thus the artifacts) based on the type of relations. For example, the parent table of a one-to-many relation has higher level than the child table of this relation, and all tables without parents are assigned to the highest level. We can allow a user to limit the scope to a certain level of artifacts.

It is also very interesting to investigate and allow hierarchy within an artifact and within an artifact-centric model. Thus, we can use sub-artifacts to model document lines which can be included within the artifacts. For example, the artifact \textit{Sales Order} can include the sub artifacts \textit{Sales Lines} within its life-cycle. We can group several artifacts and aggregate the interactions between artifacts into the interactions between the groups of artifacts. Having such a hierarchy might allow us to create an overview of an entire ERP system, while still be able to zoom into the details of life cycle of each artifact.

\textbf{Interaction Discovery}

Regarding the interaction identification, several improvements can be done. First, we can extract multiple type level interactions between two artifacts (by simply assigning each of these interactions a different key in the XES event log), while adding the functionality in the InteractionMiner to select the specific type level interactions to investigate. This extension will increase the usability of the approach and allow users and researchers to investigate multiple type level interactions at the same time without re-extract event logs.

Another very interesting extension is to merge the event logs of multiple artifacts to overcome the limitation of the current interactions which only express the (possible) causal relations between two artifacts. Take the OTC example and the model shown in Figure 1.7, if an invoice is created before a related delivery and also before the creation of the order related to the delivery, we will only be able to identify the unusual flow from the creation of invoice to the creation of delivery. We will not be able to identify this ordering between the three artifacts: first the creation of invoice, then the creation of order, followed by the creation of delivery. Therefore, we can merge the orders, deliveries and invoice based on the interactions to obtain the event log, which has a notion of a case as the combination of these three documents.

An important remark is that it is also possible to merge merge multiple event logs to obtain an event log of a large artifact, which is usually obtained by using the traditional log. For example, we can use the trace level interactions to merge the (multiple) related \textit{Delivery} traces with the \textit{Sales Order} trace.

\textbf{Process Analyses and Visualization}

The artifact-centric models we visualized in ProM used the default functions (e.g. regarding layout) provided by the ProM framework. One can use more sophisticated visualization techniques such as the edge bundling techniques [15] to improve the understandability of the model. The interviews
conducted with the advisors have also shown that using message sequence chart to visualize individual cases and their related interactions might be interesting.

As the results of the case studies have shown, to really use the discovered artifact-centric model for detecting unusual flow, more filters based on domain knowledge might be necessary to distinguish the real unusual flows from the allowed unusual flows. For example, in the case study of the Oracle PA process, we found a task which was completed before two related expenditures were ended. However, this unusual flow from Expenditure Ending to Task Completed is allowed if they are within the same week. Thus, to distinguish these cases from the allowed cases, domain knowledge and logic might have to be build into the model.

So far, we used the discovered model to identify unusual flows. It would be interesting to apply the conformance checking techniques proposed by D. Fahland et al. [12] on the artifact-centric model discovered by our approach.

**Functionality of the Tools**

A limitation of the XTract 2 tool is still the performance of log extraction, which is also discussed by E. Nooijen [21] since we reused the XTract approach.

For the SAP case study, we had to extract two version of event logs of the same artifact: a simple version and a comprehensive version. We used the simple version to obtain a better overview. However, this can also be realized by providing transition filtering function in the InteractionMiner tool. Also, being able to modeling, filtering, highlighting or comparing artifact-centric models based on attributes (e.g. different business units, different time frame) would be very useful for conducting process analyses.
Bibliography


Appendix A

OTC Example

In this appendix, we introduce the OTC example which is simplified from the table structure used by the SAP Order to Cash (OTC) process. The example is named the OTC example and used throughout this thesis. The relational data structure shown in Figure A.1 consists of four tables, Sales Documents (SD), Deliveries Document (DD), Billing Document (BD) and the Document Changes. The content of the four tables are shown in Figure 1.1 with references.

- The SD table contains two sales orders (i.e. S1 and S2) and a return order (i.e. S3), and the Document type column indicates two types of documents, Sales Order and Return order. The SD table is a simple version of the sales documents table VBAK within SAP.
- The DD table contains three delivery documents (i.e. D1, D2, D3) and a return delivery document (i.e. D4), and the Document type column indicates two types of documents, Delivery and Return delivery.
- The BD table contains two invoice documents (i.e. B1, B2), and the Document type column indicates one type of documents, Invoice.
- The DOCUMENT CHANGES table (Changes) store all changes related to different document (the column Reference id is a foreign key referring to the other three tables). The first three records is related to the sales order S1, and the fourth record is related to the invoice B2. The column Date changed indicates the date when the change is done, and the column Change type indicates the type of the change.

The four references between the tables SD, DD, BD, and changes are displayed in Figures A.1 and 1.1.

- The SD table has a reference $F_1$ with itself, i.e. the foreign key column Reference id in the SD table refers to the primary key column SD id in the SD table.
- A reference $F_2$ between the SD table and the DD table is found, i.e. the foreign key column Reference SD id in the DD table refers to the primary key column SD id in the SD table.
- There is a reference $F_3$ found between the BD table and the DD table, i.e. the foreign key column Reference BD id in the DD table refers to the primary key column BD id in the BD table.
- The reference $F_4$ is found between the table CHANGES and the other three documents tables. The column Reference id is a foreign key referring to the column SD id, the column DD id or the column BD id depending on the value in the column Table name of the table CHANGES.
Figure A.1: Data schema of the OTC example

An arc represents a reference, where the source of arc is the parent table, and the target of arc is the child table.
Appendix B

Mapping Creation and Log Extraction

In this appendix, we show an example of a (log) mapping created based on the artifact Sales Order in Figure B.1. The new data structure with respect of the mapping is shown in Figure B.2.
APPENDIX B. MAPPING CREATION AND LOG EXTRACTION

Figure B.2: Mapping data structure

Reused the mapping data structure of XTract and extended with the class GeneralMappingProperty (colored blue)
Appendix C

Case Studies

C.1 Case I - SAP OTC process

In this appendix, we show the result related to the SAP case study. Figure C.1 shows the primary keys we used for the SAP Order to Cash process, and Figure C.2 shows the foreign keys we used.

<table>
<thead>
<tr>
<th>Table</th>
<th>Primary keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>BKPF</td>
<td>Mandt, BUKRS, BELNR, GJAHR</td>
</tr>
<tr>
<td>bsad</td>
<td>MANDT, BUKRS, KUNNR, UMSKS, UMSKZ, AUGDT, AUGBL, ZUONR, GJAHR, BELNR, BUZEI</td>
</tr>
<tr>
<td>cdhdr</td>
<td>MANDANT, OBJECTCLAS, OBJECTID, CHANGENR</td>
</tr>
<tr>
<td>bsid</td>
<td>MANDT, BUKRS, KUNNR, UMSKS, UMSKZ, AUGDT, AUGBL, ZUONR, GJAHR, BELNR, BUZEI</td>
</tr>
<tr>
<td>cdpossmall</td>
<td>MANDANT, OBJECTCLAS, OBJECTID, CHANGENR, TABNAME, TABKEY, FNAME, CHNGIND</td>
</tr>
<tr>
<td>LIKP</td>
<td>MANDT, VBELN</td>
</tr>
<tr>
<td>LIPS</td>
<td>MANDT, VBELN, POSNR</td>
</tr>
<tr>
<td>VBAK</td>
<td>MANDT, VBELN</td>
</tr>
<tr>
<td>VBAP</td>
<td>MANDT, VBELN, POSNR</td>
</tr>
<tr>
<td>VBRK</td>
<td>MANDT, VBELN</td>
</tr>
<tr>
<td>VBRP</td>
<td>MANDT, VBELN, POSNR</td>
</tr>
<tr>
<td>cdpos</td>
<td>MANDANT, OBJECTCLAS, OBJECTID, CHANGENR, TABNAME, TABKEY, FNAME, CHNGIND</td>
</tr>
</tbody>
</table>

Figure C.1: SAP OTC tables - primary keys
### Figure C.2: SAP OTC tables - foreign keys

<table>
<thead>
<tr>
<th>Parent table</th>
<th>Parent columns</th>
<th>Child table</th>
<th>Child columns</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBAK</td>
<td>mandt,vbeln</td>
<td>VBAP</td>
<td>mandt,vbel</td>
<td>NULL</td>
</tr>
<tr>
<td>VBAK</td>
<td>mandt,vbeln</td>
<td>VBAP</td>
<td>mandt,vbel</td>
<td>NULL</td>
</tr>
<tr>
<td>VBAF</td>
<td>mandt,vbeln,postr</td>
<td>VBAP</td>
<td>mandt,vbel,vpos</td>
<td>NULL</td>
</tr>
<tr>
<td>VBAF</td>
<td>mandt,vbeln,postr</td>
<td>VBAP</td>
<td>mandt,vbel,vpos</td>
<td>NULL</td>
</tr>
<tr>
<td>LIPS</td>
<td>mandt,vbeln</td>
<td>VBAP</td>
<td>mandt,vbel,vpos</td>
<td>NULL</td>
</tr>
<tr>
<td>LIPS</td>
<td>mandt,vbeln</td>
<td>VBAP</td>
<td>mandt,vbel,vpos</td>
<td>NULL</td>
</tr>
<tr>
<td>VBAP</td>
<td>mandt,vbeln,postr</td>
<td>VBAP</td>
<td>mandt,vbel,vpos</td>
<td>NULL</td>
</tr>
<tr>
<td>VBAP</td>
<td>mandt,vbeln,postr</td>
<td>VBAP</td>
<td>mandt,vbel,vpos</td>
<td>NULL</td>
</tr>
<tr>
<td>VBAP</td>
<td>mandt,vbeln,postr</td>
<td>VBAP</td>
<td>mandt,vbel,vpos</td>
<td>NULL</td>
</tr>
<tr>
<td>VBAP</td>
<td>mandt,vbeln,postr</td>
<td>VBAP</td>
<td>mandt,vbel,vpos</td>
<td>NULL</td>
</tr>
<tr>
<td>VBAP</td>
<td>mandt,vbeln,postr</td>
<td>VBAP</td>
<td>mandt,vbel,vpos</td>
<td>NULL</td>
</tr>
<tr>
<td>CDHDR</td>
<td>mandant,objectclas,objectid,chanenr</td>
<td>COPOS</td>
<td>mandant,objectclas,objectid,chanenr</td>
<td>NULL</td>
</tr>
<tr>
<td>VBRK</td>
<td>mandt,vbeln</td>
<td>COPOS</td>
<td>mandant,objectid</td>
<td>&lt;child&gt;.objectclas = 'FAKTBELEG'</td>
</tr>
<tr>
<td>VBRK</td>
<td>mandt,vbeln</td>
<td>COPOS</td>
<td>mandant,objectid</td>
<td>&lt;child&gt;.objectclas = 'FAKTBELEG'</td>
</tr>
<tr>
<td>VBAK</td>
<td>mandt,vbeln</td>
<td>COPOS</td>
<td>mandant,objectid</td>
<td>&lt;child&gt;.objectclas = 'VERKBELEG'</td>
</tr>
<tr>
<td>VBRK</td>
<td>mandt,vbeln</td>
<td>COPOS</td>
<td>mandant,objectid</td>
<td>&lt;child&gt;.objectclas = 'VERKBELEG'</td>
</tr>
<tr>
<td>VBRK</td>
<td>mandt,vbeln</td>
<td>COPOS</td>
<td>mandant,objectid</td>
<td>&lt;child&gt;.objectclas = 'VERKBELEG'</td>
</tr>
<tr>
<td>VBRK</td>
<td>mandt,vbeln</td>
<td>COPOS</td>
<td>mandant,objectid</td>
<td>&lt;child&gt;.objectclas = 'VERKBELEG'</td>
</tr>
<tr>
<td>BKPF</td>
<td>mandt,awkey</td>
<td>BKPF</td>
<td>mandt,awkey</td>
<td>&lt;child&gt;.awtyp = 'VBRK'</td>
</tr>
<tr>
<td>BSAD</td>
<td>mandt,beinr,BUKRS,GJAHR</td>
<td>BSAD</td>
<td>mandt,beinr,BUKRS,GJAHR</td>
<td>NULL</td>
</tr>
<tr>
<td>BSAD</td>
<td>mandt,beinr,BUKRS,GJAHR</td>
<td>BSAD</td>
<td>mandt,beinr,BUKRS,GJAHR</td>
<td>NULL</td>
</tr>
</tbody>
</table>