MASTER

Artifact-centric process analysis
process discovery in ERP systems

Nooijen, E.H.J.

Award date:
2012

Eindhoven University of Technology
Department of Mathematics and Computing Science

Master's Thesis on

Artifact-Centric Process Analysis

Process discovery in ERP systems

E.H.J. (Erik) Nooijen

In partial fullfilment of the requirements for the degree of

Master of Science
in Business Information Systems

Supervisors:
dr.ir. B.F. (Boudewijn) van Dongen (TU/e – W&I – AIS)
dr. D. (Dirk) Fahland (TU/e – W&I – AIS)

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 257593

Project Acronym	ACSI	
Project Title	Artifact-Centric Service Interoperation	
Project Number	257593	
Workpackage	WP3 Observation-based techniques and tools	
Lead Beneficiary	TU/e and UTARTU	
Editor(s)	Erik Nooijen	TU/e – AIS
Reviewer(s)	Boudewijn van Dongen	TU/e – AIS
	Dirk Fahland	TU/e – AIS
	George Fletcher	TU/e – DH
	Rick Hull	IBM
Actual Delivery Date	27-4-2012	
Version	1.0	

“If I have seen a little further it is by standing on the shoulders of Giants.”

Isaac Newton, 1676

Preface

Suppose you are building a house in the 1920's. You are using the same traditional tools as your mentor did: a hammer, a screwdriver and a chisel. They're simple tools, but due too your skill with them you can do a great job.

Now somebody approaches you for a demonstration: They have a new tool called an electric drill that will allow you to do your work a lot faster. The demonstration is impressive and you decide to try the electric drill yourself. Back at the house you are building you realise there is a problem though: the house does not have electricity. Of course you could set up electricity, but that would take a few weeks. That isn't necessary to finish the house, though: if you would just use the traditional tools you can finish it in a few weeks anyway...

The current Business Process Analysis (BPA) field is in some ways similar to the construction field in the 1920's. Automated support for BPA is usually poor [1], [2] and as a result the human (skill) factor is highly important for succesfull process analysis [3]. Process mining aims to aid the business analyst, but it suffers from a major practical drawback: the assumption that data is available in a pre-defined format that doesn't match real process data. This especially true for data in the ERP systems that support many primary processes in large companies: ERP systems were not build with concepts like "trace" and "event" in mind. This is a pity since the combination of process mining and ERP systems should be a golden combination[1]. Thus process mining is to the business analyst now what an electric drill was to a carpenter in the 1920's: Incredible potential, but in many cases not usefull since too much work has to be done up front.

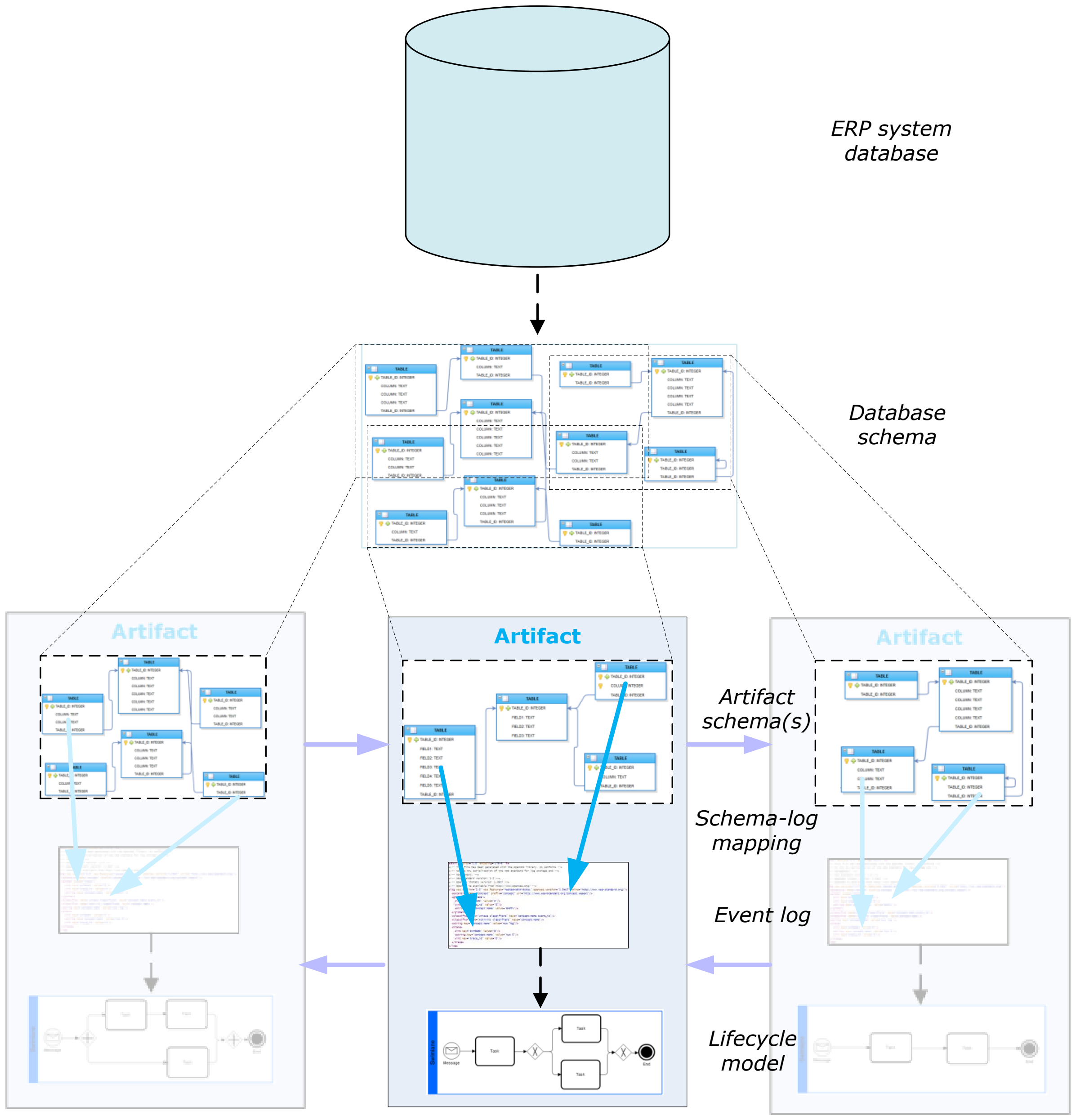
Acknowledgements

This thesis is the final part of my master at the TU/e. It's been quite a few years and it would have been a lot less pleasant without the support of many others. First and foremost of these is my wife Jori, who stood by me the entire time and kept everything going at home when I was busy with some project again. I look forward to the next period when we'll have more spare time together. Next I would like to thank the staff at the TU/e for their great attitude towards students, especially Boudewijn who in all these years always spent more time than available to him when I had another question. Finally I would like to thank some of the stable factors in the ever-changing group of students that accompanied me during these last years. Danny, Joos, it's always been a pleasure working with you guys.

[1] Appendix A gives a reasoning why this is the case

Abstract

An approach is presented that decomposes the task of creating an artifact-centric description of an ERP system into pieces for which support can be provided. For each of these pieces techniques are provided that can aid in finding the relevant information, including a set of techniques that is the first to aid in the creation of a mapping between a database and an event log that can be used for process mining. The approach is evaluated empirically showing that the method can be used, but that there are also bounds to several of the presented techniques.



Artifact-centric process analysis approach

Executive summary

Business Process Analysis can be done more efficiently by using information stored in Enterprise Resource Planning (ERP) systems. ERP systems likely contain all data that is relevant for a business process, since they are data driven and meant to support complete business processes, even if they span organizational boundaries.

Previous approaches to use this data assumed that business processes can be seen in isolation. Given the collaborative nature of ERP systems this assumption usually does not hold in environments where these systems are used. At points of interaction between processes defining what was part of the process was hard and time consuming in previous approaches as a result.

A new process analysis approach provides a way to model the collaborative environment in which ERP systems operate. This requires looking at business processes as a set of interacting business entitites called artifacts. Each of these artifacts can be described by an information schema and a non-trivial lifecycle. These descriptions of each artifact can be created efficiently using the approach shown in the figure to the left.

Since the schema describing an *ERP systems database* is often incomplete first schema extraction techniques are used to reconstruct this *database schema*. The database schema is then partitioned into self-contained *artifact schemas*, thereby identifying each artifact.

For each artifact the lifecycle can subsequently be discovered efficiently using process discovery techniques. Process discovery is a set of techniques that generates process models using only an event log as input; an *event log* is a collection of recorded events for one specific business process. Thus a *mapping* needs to be created between the artifact schema and the event log that is required. This allows the generation of the event log and subsequently the *lifecycle model* of the artifact.

A variety of techniques can be used to automate the steps described above. Empirical evaluation of these techniques on artificial and large real-life datasets shows that the approach can be used, but scalability to large datasets is limited for a number of techniques. The lifecycles discovered by the approach are reasonably accurate however.

Table of contents

Preface i
Acknowledgements i
Abstract ii
Executive summary iii
Table of contents iv
Chapter 1 – Introduction 1
1.1 Business process analysis 1
1.1.1 Business processes 1
1.1.2 Process models 1
1.1.3 Process Mining 1
1.2 Enterprise resource planning (ERP) systems 2
1.2.1 ERP systems and process mining success factors 2
1.3 Traditional process analysis in the context of ERP systems 3
1.3.1 ERP system modelling 3
1.3.2 Process mining case studies 3
1.3.3 Processes in isolation? 4
1.4 Artifact-centric process analysis in the context of ERP systems 4
1.4.1 Artifacts 4
1.4.2 Research question 5
1.4.3 Project goal and scope 6
1.5 Running example: The CD shop 7
1.5.1 Introduction 7
1.5.2 Core process: selling CD's 7
1.5.3 Database schema 8
1.5.4 Relevant artifacts 10
1.6 Outline 12
Chapter 2 – Overall method 13
2.1 Schema extraction 14
2.2 Identify artifact schemas 14
2.3 Create schema-to-log mapping 15
2.4 Generating traces 17
2.5 Apply process discovery techniques 17
Chapter 3 – Artifact schema identification 19
3.1 Relational databases 19
3.2 Schema extraction 20
3.2.1 Domain extraction 20
3.2.2 Primary key extraction 22
3.2.3 Foreign key extraction 24
3.3 Identify artifact schemas 27
3.4 Conclusion 30
Chapter 4 – Artifact lifecycle identification 31
4.1 Related work: Event log extraction 31
4.2 Create schema-to-log mapping 32
4.2.1 Overal approach 33
4.2.2 Identifying event types 35
4.2.3 Assign attributes 35
4.2.4 Creating the mapping 37
4.2.5 Mapping the CD shop quote artifact 39
4.3 Generating traces 40
4.3.1 Evaluation of previous approaches 41
4.3.2 Approach 41
4.4 Event log based artifact lifecycle discovery 42
4.4.1 Related work: Control flow discovery 42
4.4.2 Approach 43
4.5 Conclusion 44

Chapter 5 – Empirical evaluation 45
5.1 Dataset descriptions....45
5.2 Artifact schema identification 46
5.2.1 Schema extraction 46
5.2.2 Identify artifact schemas 49
5.3 Artifact lifecycle identification 50
5.3.1 Create schema to log mapping 50
5.3.2 Generating traces 50
5.3.3 Verifying generated event logs 51
Chapter 6 – Conclusions and future work....53
6.1 Conclusions....53
6.2 Future work 53
6.2.1 Further evaluation of the techniques in each step 53
6.2.2 Global optimizations....53
6.2.3 Schema extraction when data contains errors....54
6.2.4 Human interaction 54
6.2.5 Analyzing interactions between artifacts 54
6.2.6 Dropping assumptions on the data 54
Bibliography 55
Table of tables 62
Table of formulas 62
Table of figures 63
Appendix A : Why process mining of ERP systems?....65
A.I Business Process Management 65
A.II Organizational context....65
A.II.I Contingency theory 65
A.II.II Types of organizations....65
A.II.III Suitability of BPM and ERP for organizational types 66
A.III Success factors 66
A.IV Process mining of ERP systems creates synergy....66
Appendix B : Notations used 67
Appendix C : Translating to XESame 69
Appendix D : More advanced algorithms....73
D.I Efficiently selecting instance and event tables 73
D.II Alternative main table selection 74
Appendix E : Quote schema-to-log mapping 77
Appendix F : Prototype implementation 83
F.I Overview 83
F.II Data transfer 85
F.III Artifact schema identification 89
F.III.I Schema extraction 89
F.III.II Identify artifact schemas 95
F.IV Artifact lifecycle identification 96
F.IV.I Create schema to log mapping 96
F.IV.II Event log generation 98
F.V License(s) 99
Appendix G : Lifecycle verification settings 101
G.I Order 101
G.II Quote....101

Chapter 1 – Introduction

Business Process Analysis (BPA) is often very time-consuming, but that should not be the case. Analysts use traditional methods such as interviews to obtain information about key business processes, while a large amount of information is stored in systems used in organizations. An important class of systems in this context are Enterprise Resource Planning (ERP) systems, since these are used to execute many important business processes in larger companies. More often than not this information is not readily available however, so techniques are needed to easily extract process information from these systems. This requires a new way of looking at business processes that more closely corresponds to reality.

First an overview of key concepts used throughout this report is given. Then the main research problem is introduced and a direction for the solution is presented. Finally an overview is given of the material presented in the remainder of this report.

1.1 Business process analysis

Business process analysts use a variety of techniques to analyze business processes. These processes are often described using process models. Although most analysts use traditional methods such as interviews, process mining is starting to become an alternative.

1.1.1 Business processes

Business processes were first brought to the attention of many by Hammer in 1993 [4]. A recent paper described a business process as "… a complete, dynamically coordinated set of activities or logically related tasks that must be performed to deliver value to customers or to fulfill other strategic goals" [5], which is an extension of the definition given at that time.

Sometimes we want to refer to a single occurance of a business proces (e.g. the handling of a single customer order for the customer order fulfillment process): this will be called a ***process instance***.

1.1.2 Process models

Central in BPA are process models that show the business process at an abstract level. A variety of model types is used, such as control flow models (that show which tasks occur in which order), decision schema's (that show what the criteria for a decision are) and organizational models (that show the organizational hierarchy or interactions between people). The notations used to draw these models varies widely; Example notations that are used to draw control flow models include BPMN [6], petri nets [7] and Event Driven Process Chains (EPC's, [8]).

1.1.3 Process Mining

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…" [9]. It is a relatively young research discipline, which is starting to draw significant attention from the industry. The basis of process mining is using information that is recorded in software systems in the form of event logs for process analysis. The most important advantages of this approach over traditional methods are a large reduction of the time required for such analysis and a certainty that facts are presented in the results (instead of opinions).

An ***event log*** is a collection of recorded events for one specific business process. It consists of a set of traces for that process. Each ***trace*** is a list of events that occurred for a single process instance. A trace is ordered: an event that occured before another event should be earlier in the trace as well. An ***event*** is something that happened at some point in time. Events are instanteneous: they end at the same time that they started. Therefore a prolonged activity consists of several events, for example the start and end of the activity.

Process discovery is the set of techniques that generates process models using only an event log as input. Since event logs are a record of real behaviour that occurred, the resulting process models always reflect reality.

1.2 Enterprise resource planning (ERP) systems

ERP systems are software systems that support the optimal usage of all resources in an organization. The underlying idea of ERP is described in the 11th edition of the APICS dictionary as a "framework for organizing, defining, and standardizing the business processes necessary to effectively plan and control an organization so the organization can use its internal knowledge to seek external advantage" [10]. Adoption of ERP systems is high: 74% of large manufacturing companies used an ERP system in 2002 [11], 80% of Fortune 500 companies used an ERP system in 2004 [12] and a rising 56% of small and medium enterprises in Europe used an ERP system in 2009 [13].

The traditional ERP system is data-centric and function oriented [14]. The support provided by the system is done through a shared data model that is used by a variety of functions, thus ensuring that each function uses the same version of the truth. This data is usually stored in a single relational database containing thousands of tables [15–17].

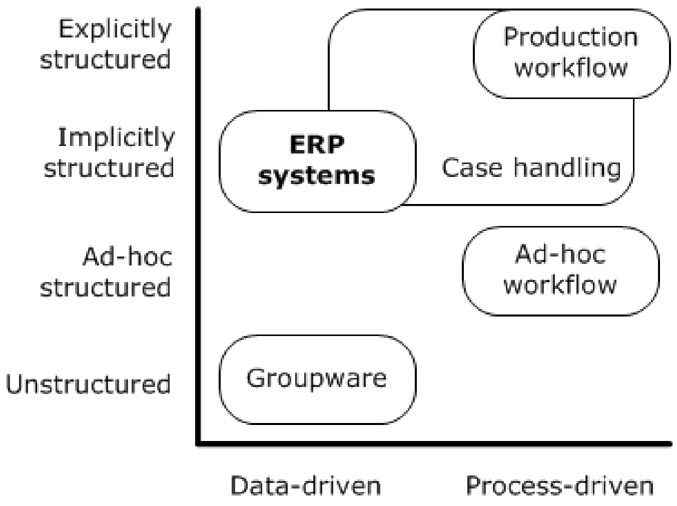


Figure 1: Process support systems (adapted from [18])

ERP systems were build with structured processes in mind; Prefered models are reflected in so-called reference models that describe both the underlying data and process models [14]. Hence business processes are supported, but only implicitly: the processes are hidden in the code of the system. Figure 1 shows the positioning of ERP systems when compared to other process support systems.

1.2.1 ERP systems and process mining success factors

For process mining to be succesful event data of sufficient quality is required. The Process Mining Manifesto states that the minimum requirement is that events are automatically recorded and that there should be some sort of guarantee that recorded events match reality [9]. As stated in the manifesto this is the case for ERP systems. An additional quality measure for event logs is completeness: no events should be missing for the process that is being analyzed [9]. ERP systems were not build with event logs in mind, so there is no guarantee that this is the case for these systems, but ERP systems do increase the likelyhood of completeness. Two factors contribute to this:

- ERP systems are meant to aid in cross-departemental activities. The idea is that business processes can be handled in a single system (so there is only one version of the truth).
- ERP systems are data driven by nature, thus information that is used during the process will usually be stored in the ERP system.

1.3 Traditional process analysis in the context of ERP systems

Since many important business processes are supported by ERP systems it is rather surprising that very little work has been done on process analysis in the context of these systems. Related work on the modelling of ERP systems describes what should be taken into account for analysis in the context of these systems. In addition to this a variety process mining case studies have been executed focussing on ERP systems. It can be argued that these approaches neglect a fundamental property however.

1.3.1 ERP system modelling

The authors of [19] describe that a notation used to model an ERP system should be able to describe both data and control flow aspects. In addition to this they describe a manual approach to modelling an ERP system and note that this is very laborious task [19]. Similarly, the reverse engineering work given in [20] describes that both data and processes should be transferred, but does not describe how this could be done.

1.3.2 Process mining case studies

Current process discovery techniques assume that event log information is available in a predefined format. In this format a log consists of a set of traces (with each trace corresponding to a specific case or process instance). Each trace consists of an ordered multiset of events that have taken place; For each event the event type should be known.

In practice event log information is not readily available in this form in most information systems. Previous research has shown that extracting log information from ERP systems is a time consuming process as explained below. As a result one of the major advantages of process mining (reduced analysis time) is completely offset by the time required for preprocessing of information.

The only ERP systems that were studied were SAP [15], [16], [21–23] and PeopleSoft [17]. A variety of approaches are tested to extract event logs from these systems, but the majority of these fail because no instance identifier is available for the events that can be extracted. For the specific use cases of the SAP case studies event log extraction from the SAP database is succesful, although selection of the correct tables is laborious. Usually an event is identified by a column with a timestamp, but the exact event type may depend on the value of another column (in the same table) [21], [22].

The case study on PeopleSoft reports that no general approach to extract event logs from PeopleSoft is possible, although it appears to be possible to manually select the correct tables that contain the customer billing information; the main issue is the absence of foreign keys in the database [17]. An additional issue reported for Peoplesoft is that information about applications started for the execution of an action is only stored until the execution of the application is finished [17]. The absence of foreign keys is also repeatedly reported for the database of SAP, although Piessens shows that the foreign key information can be retrieved from the SAP system in a rather laborious way [15].

A challenge encountered in all case studies is divergence and convergence. Both are best explained through an example: Suppose we are dealing with an online CD shop, where customers order CD's from the web. The shop collects these orders on a daily basis and then orders the required CD's at its suppliers. When looking at the customer order fulfillment process this provides two interesting points:

- The shop will place one orders at a supplier for multiple customer orders. Thus an event can be associated to multiple process instances. This situation is called ***convergence***.
- The shop will place orders at multiple suppliers for a single customer order if not all CD's can be bought from the same supplier. Thus there will be multiple similar events for a single process instance. This situation is called ***divergence***.

To handle divergence Piessens uses foreign key information to suggest instance identifiers by selecting primary key fields that are (indirectly) referenced by all event tables. Piessens also suggests that it would be a significant improvement if a method would be developed that automatically discovers event types by focussing on timestamps.

1.3.3 Processes in isolation?

Many existing process analysis methods (including the process mining case studies discussed above) assume that processes can be seen in isolation [24] as visualized in Figure 2. Given the collaborative nature of ERP systems this assumption usually does not hold in environments where these systems are used. This causes problems when convergence occurs, because convergence by definition means that interaction between multiple process instances should be taken into account. The main issue is that it becomes unclear what defines a process (instance), i.e. one cannot define precisely what is part of the process and what is not. Because of this a different process instance defintion tends to be chosen for each situation that is analyzed.

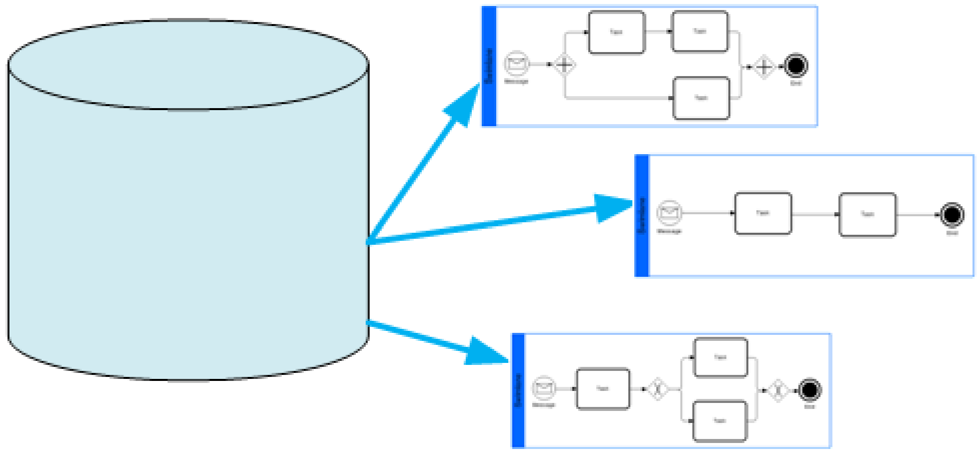


Figure 2: Traditional view on processes

The unclarity of the definition of a process instance explains the problems encountered in the case studies on process mining in ERP systems with convergence, divergence and the identification of instance identifiers. In addition to this the unclarity also makes it hard to identify the relation between the data and the control flow, since this will vary based on the process instance definition chosen to explain a specific situation. Thus a new process analysis approach is required that takes the interaction between processes into account.

1.4 Artifact-centric process analysis in the context of ERP systems

1.4.1 Artifacts

Artifacts are business entities described by both an information model and a non-trivial lifecycle [25], [26]. An example would be customer orders or purchase orders. One such entity (e.g. a specific customer order) is called an ***artifact instance***.

Artifacts can be related to other artifacts. For example, in a build-to-order environment a new customer order could trigger the creation of a new purchase order. This purchase order might be related to multiple customer orders though, since a single purchase order might be used to purchase the required materials for multiple customer orders. This also implies that the lifecycles are mutually dependent on eachother: the creation of a customer order causes the creation of a purchase order, but the arrival of the goods of the purchase order trigger the further processing of the customer order.

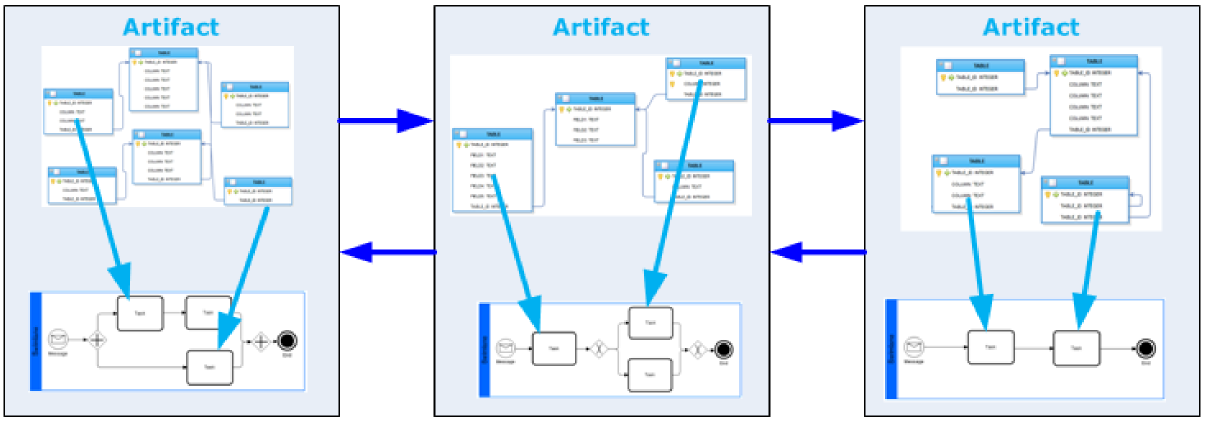


Figure 3: Artifact view on business processes

The authors of [27], [28] describe a business process analysis approach that describes an environment as a set of artifacts. Figure 3 visualizes the general idea of this approach, showing the interaction between artifacts as dark-blue arrows and the mapping between the information model and the lifecycle as light-blue arrows. In this approach the business artifacts are identified by first looking at *what data is important* and only then it is investigated *how things should be done*. The required information is gathered through the use of interviews [29], which can be time-consuming.

1.4.2 Research question

Although previous work on business process analysis in the context of ERP systems clearly shows that data, processes and interaction between processes should be taken into account, support for this is rather limited. The artifact-centric approach provides a way to describe how things work, but the interviewing approach is time-consuming. Although process mining could in theory be usefull its practical value is limited, since the required time for preprocessing offsets one of its major advantages. Thus an artifact-centric method is required that combines existing techniques to provide an efficient way of doing process analysis in the context of ERP systems, resulting in the following research question:

How can an artifact-centric description of an ERP system efficiently be derived from the systems database?

Two important aspects of artifacts are the information schema describing the data it contains and the lifecycle that describes how it functions. Therefore the research question can be decomposed into two subquestions:

1) *How can the information schema of each artifact in an ERP system be identified using the systems database?*

2) *How can the lifecycle of each artifact in an ERP system be identified using the systems database?*

1.4.3 Project goal and scope

Given the research question the goal of the project is to *create a generic, semi-automated method that generates a description of all artifacts represented in a given structured dataset, supported by a coherent set of techniques*. Note that an ERP systems database is an example of a structured dataset. This will be approached by first reconstructing the information schema of the dataset. Then the artifacts and their information schemas will be identified by partitioning the information schema of the complete dataset. Since artifact instances can then be defined clearly traditional process discovery techniques can subsequently be used to identify the lifecycle of each artifact via an event log. Figure 4 shows the general idea of this approach.

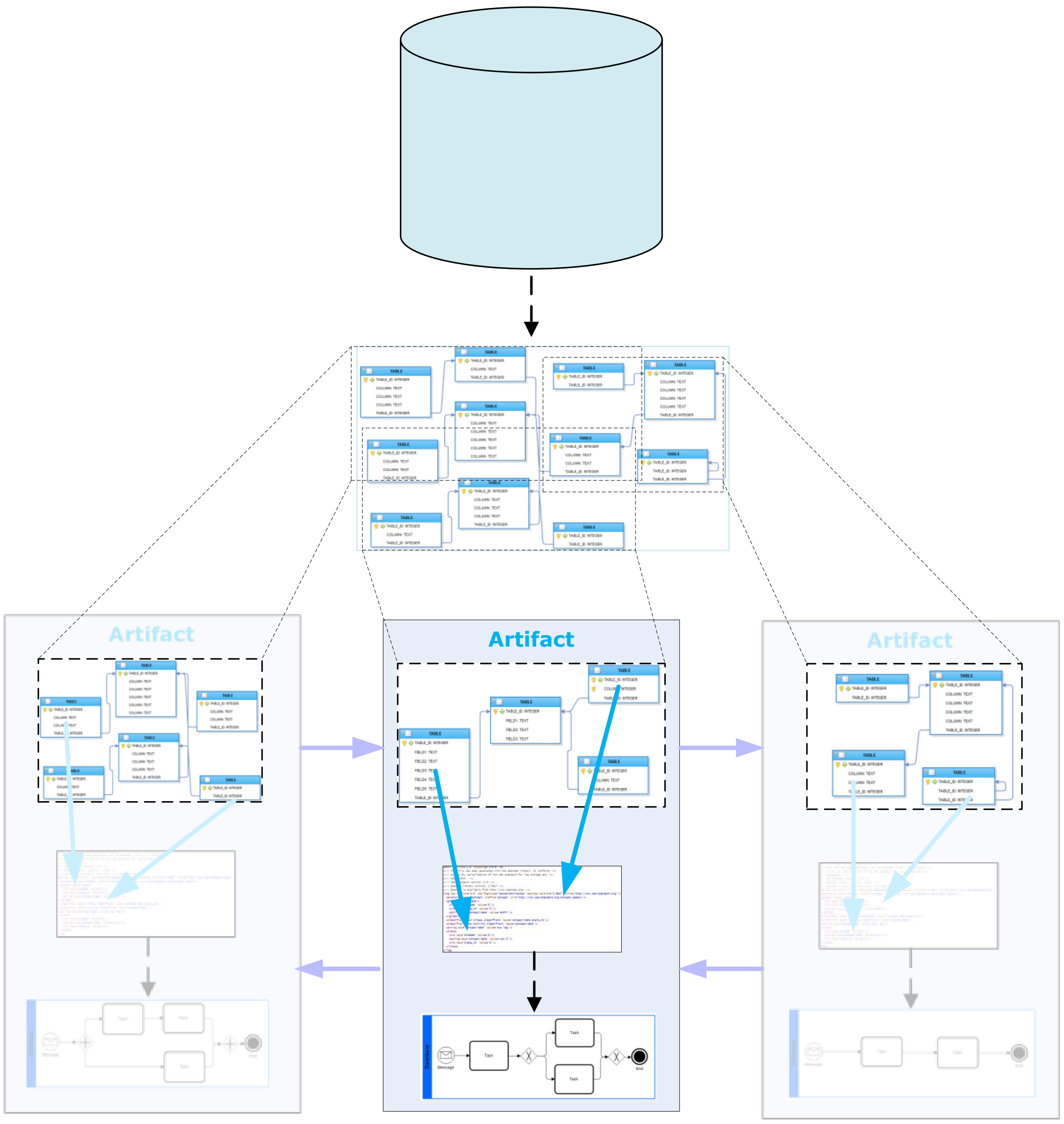


Figure 4: General idea of the approach

For practical purposes some assumptions were made to limit the scope of the project:

- The most important business processes are all lifecycles of some artifact. Thus focussing only on artifact lifecycles is sufficient. Extracting information of business processes that are not artifact lifecycles is outside of the scope of this project.
- Although the interaction between artifacts is important to fully describe their behaviour there is no previous work on (semi-)automated interaction discovery and it is beyond the scope of this project to develop such a method. Therefore the artifact lifecycle discovery will be limited to its inner lifecycle.
- The information inside ERP systems is stored in relational databases. Process analysis for ERP systems with a different way to store data may not benefit from the developed method.
- The ERP systems database(s) can be accessed directly so information can be extracted from them. Extracting information from ERP systems for which the database is not accessible is outside the scope of this project.

1.5 Running example: The CD shop

1.5.1 Introduction

In this section the CD shop is presented: a fictional company that is used as a running example throughout the report. The CD shop is a small webshop where customers can order CD's. The shop subsequently orders CD's from their suppliers to be able to deliver CD's to their customers. In the next subsections the CD shop is described in detail. First the core process of the CD shop is described. Then an overview of the database schema of the shops system is described. Finally an overview is given of the three artifacts relevant for the CD shops core business process and how these artifacts relate to this process.

In this report entity-relation (ER) diagrams will be used to describe the information model of an artifact while the lifecycle will be described using proclets [24]. ER diagrams are explained in any general textbook on databases (e.g. [30], [31]). Each ***proclet*** describes the lifecycle of an artifact, including its possible interfaces with other artifacts. A proclet system consists of several proclets and as such describes the interaction between the artifacts. For this the notion of channels is used: each *channel* describes a uni-directional message flow from one artifact to another. Each *interface* of an artifact is described by its cardinality (C) and multiplicity (M): the cardinality is the number of other artifacts to which a message is sent or received from over the interface, while the multiplicity is the number of times a message can be sent or received over a specific interface during the lifecycle of one artifact instance. Both are typically represented as "1" (exactly once), "?" (zero or once), "*" (zero or more) and "+" (once or more).

1.5.2 Core process: selling CD's

The process starts when a customer *requests* if a number of *CD*'s are available in the shop. The shop then replies with a *quote* that states which CD's can be delivered and for which price. The customer then either accepts the quote or rejects it. If it is accepted a purchase *order* is placed at the supplier for the required CD's. Note that one purchase order may be used to get the CD's required for multiple customer's quotes and that orders at multiple suppliers may be required for a single customer's quote. The CD's are then shipped by the supplier to the CD shop after which the shop pays the supplier. The CD shop subsequently *delivers* the CD's to the customer. In some cases a supplier may signal that an order cannot be delivered (as a whole); In that case a reorder may be placed at the same or a different supplier. If the CD's cannot be delivered by any supplier the customer is notified of this and the process ends. If the CD's are delivered normally the customer receives an invoice. After the customers payment is received the process ends.

1.5.3 Database schema

The CD shops database contains all data required for its core process. Figure 5 shows the relational schema of the database, while details of each table are given in Table 1.

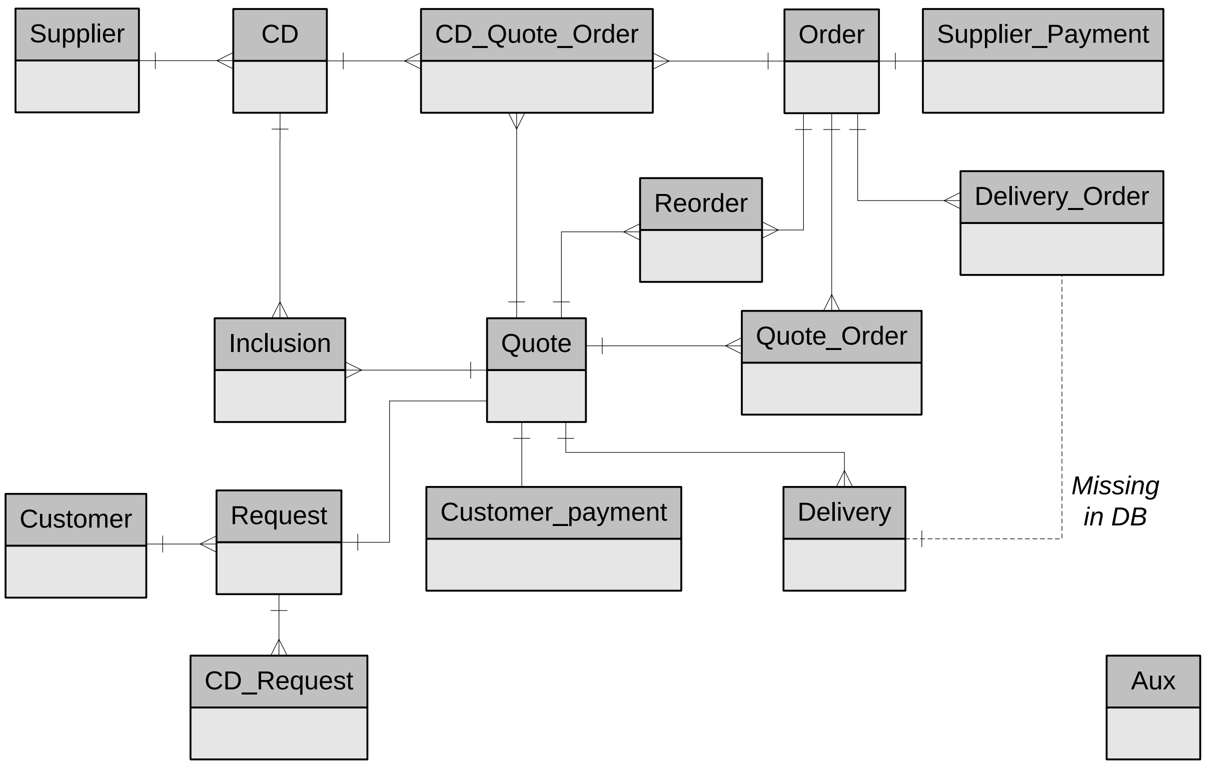


Figure 5: CD shop database schema

Although fictional, the CD shops database has some deliberate flaws that real databases also have. First of all the foreign key between the *delivery* and *delivery_order* table is missing in the database. In addition to this several columns have a datatype that is not completely correct: one *price* has a "real" datatype which should be an "integer" and two *name* columns have a "char" datatype which should be a "varchar". All of these are marked with an asterisk (*) in the table below.

Table name	*Column name*	*PK*	*Datatype*	*Data length*
Aux	Aux	Yes	Integer	
Cd	Name	Yes	Varchar	50
	Artist		Varchar	30
	Price		Real*	
	Supplier_Name		Varchar	10
Cd_Request	Request_Reqid	Yes	Integer	
	Cd_Name	Yes	Char*	50
	Quantity		Integer	
Cdquote_Order	Quote_Reqid	Yes	Integer	
	Order_Orderid	Yes	Integer	
	Cd_Name	Yes	Varchar	variable
	Quantity		Integer	
	Deliverable_Quantity		Integer	
Customer	Name	Yes	Varchar	30
Customer_Payment	Quote_Reqid	Yes	Integer	

Table name	*Column name*	*PK*	*Datatype*	*Data length*
	Date_Invoice_Issue		Timestamp	
	Date_Payment_Sent		Timestamp	
	Date_Payment_Received		Timestamp	
	Price		Integer	
Delivery	Customer_Accept_Shipment_Date		Timestamp	
	Delid	Yes	Varchar	6
	Quote_Reqid		Integer	
Delivery_Order	Delivery_Delid	Yes	Varchar	6
	Order_Orderid	Yes	Integer	
Inclusion	Cd_Name	Yes	Varchar	50
	Quote_Reqid	Yes	Integer	
	Quantity		Integer	
Order	Orderid	Yes	Integer	
	Order_To_Supplier_Date		Timestamp	
	Supplier_Notification_Date		Timestamp	
	Supplier_Shipment_Date		Timestamp	
	Opening_Date		Timestamp	
Quote	Reqid	Yes	Integer	
	Price		Integer	
	Opening_Date		Timestamp	
	Acceptance_Quote_Date		Timestamp	
	Rejection_Quote_Date		Timestamp	
	Customer_No_Deliverable_Notification_Date		Timestamp	
Quote_Order	Quote_Reqid	Yes	Integer	
	Order_Orderid	Yes	Integer	
	Adding_Date		Timestamp	
Reorder	Quote_Reqid	Yes	Integer	
	Order_Orderid	Yes	Integer	
	Reorder_Date		Timestamp	
Request	Request_Date		Timestamp	
	Reqid	Yes	Integer	
	Customer_Name		Varchar	30
Supplier	Name	Yes	Char*	10
Supplier_Payment	Order_Orderid	Yes	Integer	
	Date_Invoice_Issue		Timestamp	
	Date_Payment_Sent		Timestamp	
	Date_Payment_Received		Timestamp	
	Price		Integer	

Table 1: CD shop metadata

1.5.4 Relevant artifacts

For the CD shop's core process three artifacts are of primary importance: quotes, (purchase) orders and (physical) CD's. Below these artifacts are introduced.

In this example entity-relation (ER) diagrams are used to describe the information model of an artifact while the lifecycle is described using proclets [24]. ER diagrams are explained in any general textbook on databases (e.g. [30], [31]). Each ***proclet*** describes the lifecycle of an artifact, including its possible interfaces with other artifacts. A proclet system consists of several proclets and as such describes the interaction between the artifacts. For this the notion of channels is used: each *channel* describes a uni-directional message flow from one artifact to another. Each *interface* of an artifact is described by its cardinality (C) and multiplicity (M): the cardinality is the number of other artifacts to which a message is sent or received from over the interface, while the multiplicity is the number of times a message can be sent or received over a specific interface during the lifecycle of one artifact instance. Both are typically represented as "1" (exactly once), "?" (zero or once), "*" (zero or more) and "+" (once or more). Note that this approach provides a solution for the divergence/convergence challenge described in section A.I [24].

Quote

A quote is basically a customer order. It starts its lifecycle as a request from a customer and usually ends when it is paid for. Figure 6 shows the database schema of the quote artifact. The *quote* table contains a list of quote instances. Since all quotes start as a request technically the same could be said for the *request* table, but this table obviously has a slightly different meaning attached to it. The lifecycle of a quote instance is shown in Figure 8 as part of a proclet system that contains both the quote and order lifecycle.

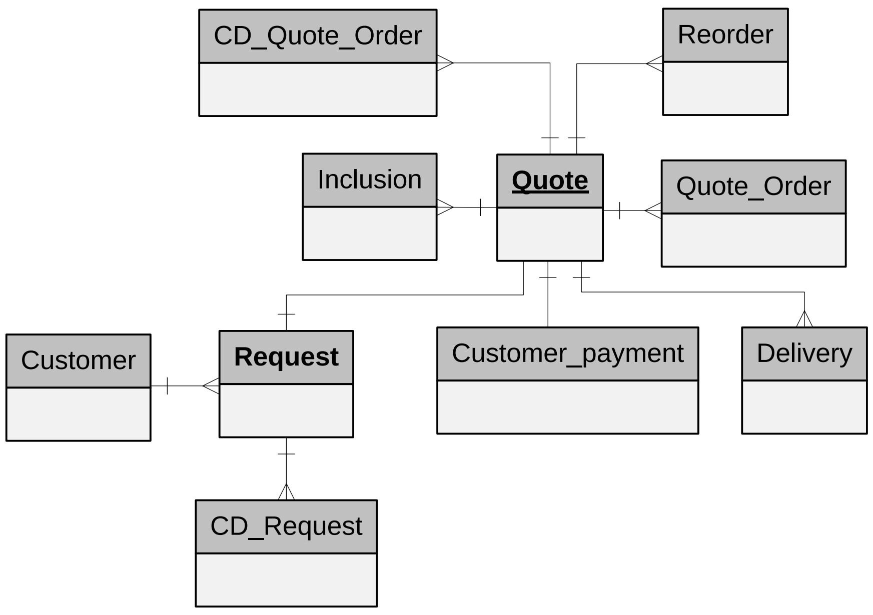


Figure 6: Quote schema

Order

An order is used to purchase CD's from a supplier. It starts its lifecycle when the CD shop requests the delivery of CD's and usually ends when the purchase order is paid for. Figure 7 shows the database schema of a the order artifact. The *order* table contains a list of order instances. The lifecycle of an order instance is shown in Figure 8 as part of a proclet system that contains both the quote and order lifecycle.

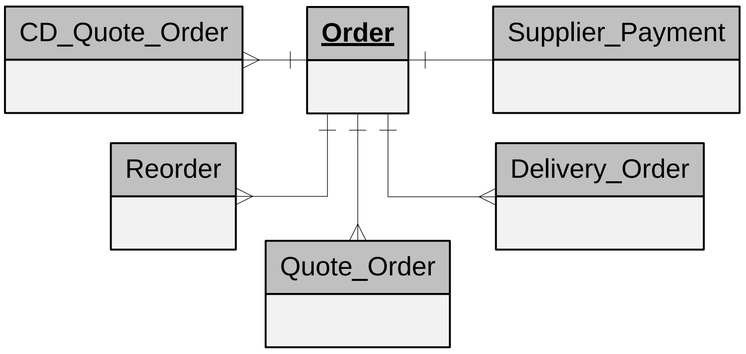


Figure 7: Order schema

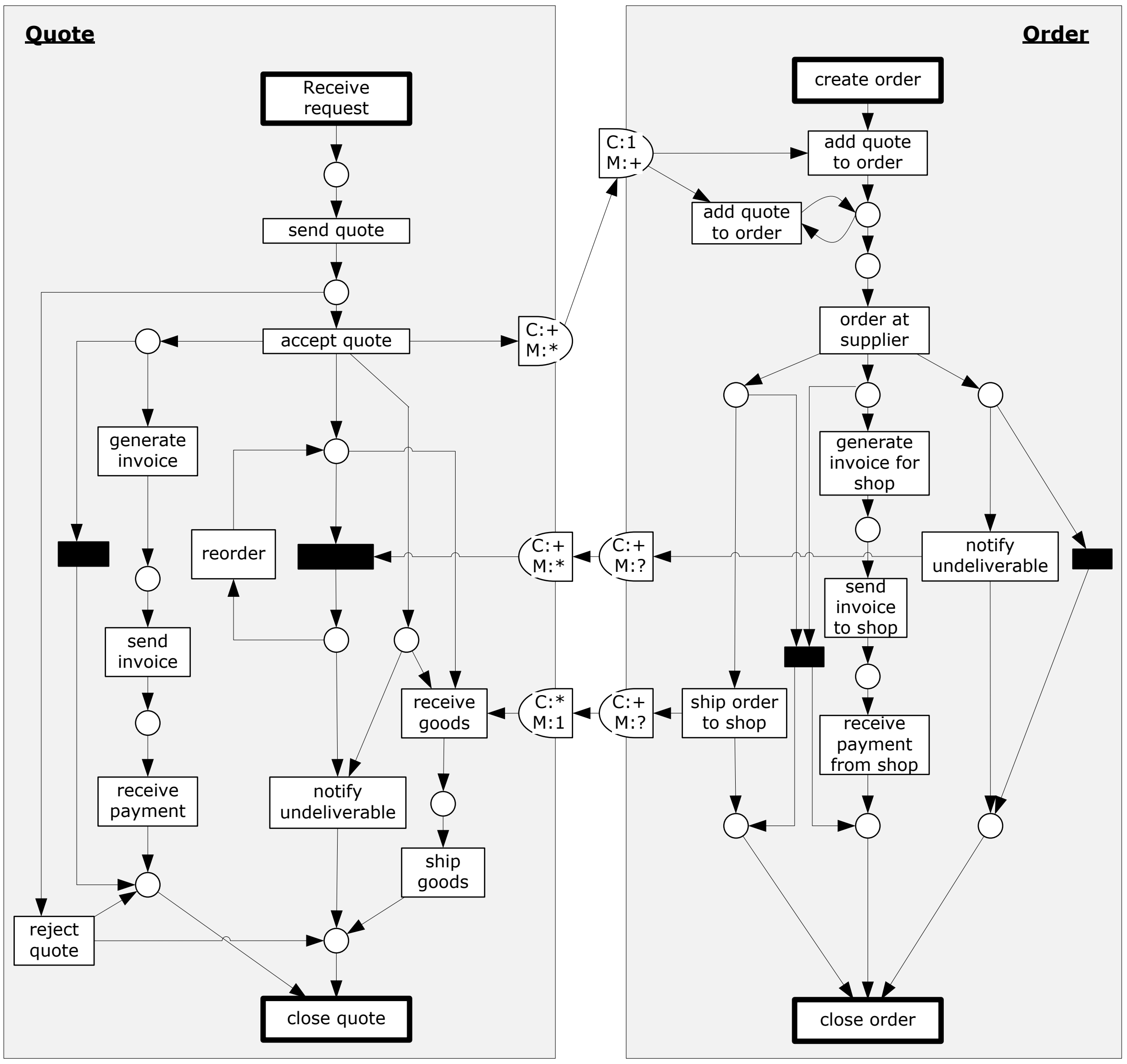


Figure 8: Quote and order lifecycles

CD

A physical CD is the object that is handled during the entire core process of the CD shop. It starts its lifecycle when it is requested by a customer and ends when it is delivered to a customer. Its lifecycle is never individual however, since the lifecycle is always shared by other CD's that were requested by the same customer and/or delivered by the same supplier. Thus one could say that a CD does not have a lifecycle of its own; It has a lifecycle, but that lifecycle is completely represented by the related *quote* and *order* artifacts.

Figure 9 shows the database schema of a physical CD. There is no table that contains a list of physical CD's in this schema, though. The *CD* table only contains the type of CD's available in the CD shop. The *inclusion* and *cd_quote_order* tables get closer to describing physical CD's: the *inclusion* table contains a list of physical CD's that are included in quote instances, while the *cd_quote_order* table contains a list of physical CD's that are included in both a quote and an order instance.

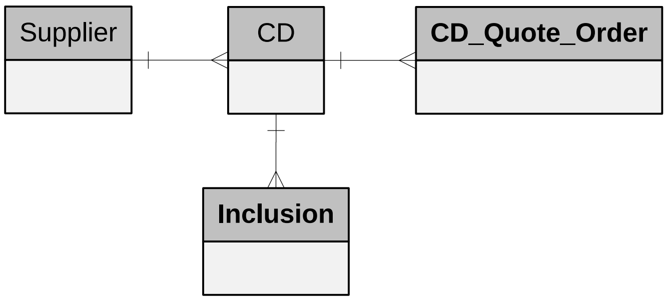


Figure 9: CD schema

1.6 Outline

The remainder of this report is structured as follows. Chapter 2 gives an overview of the proposed method followed by a detailed explaination of the artifact schema discovery part in chapter 3 and the artifact lifecycle discovery part in chapter 4. In chapter 5 the method is then evaluated experimentally. The report is concluded with ideas for future work in chapter 6. Finally Appendix B contains an overview of all notations used in this report.

Chapter 2 – Overall method

The goal of the method is to generate a description of all artifacts represented in any given structured dataset that contains artifact data and event information (possibly among other things). Figure 10 shows the steps required to achieve this.

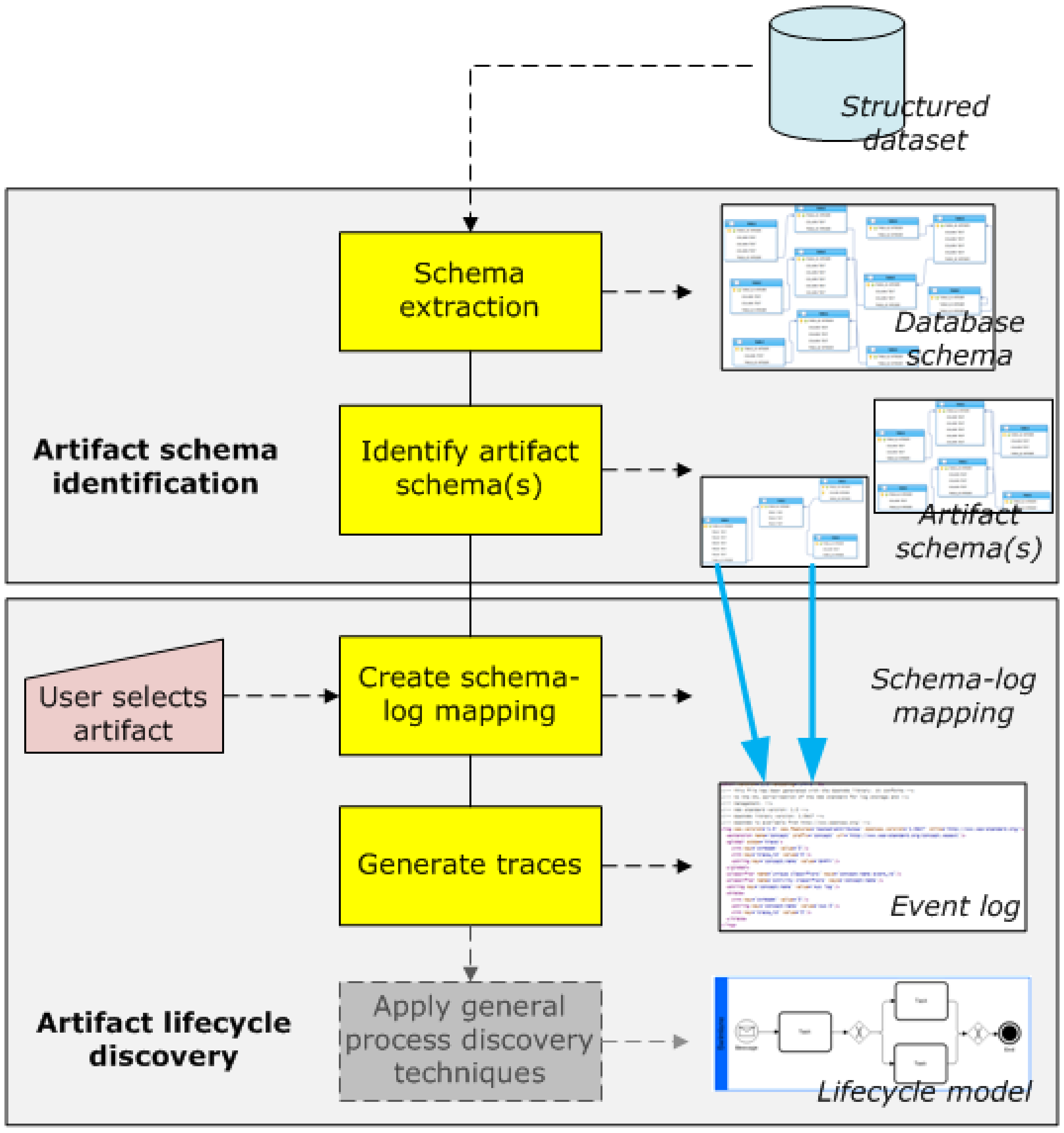

Structured dataset
Schema extraction
Database schema
Artifact schema identification
Identify artifact schema(s)
Artifact schema(s)
User selects artifact
Create schema-log mapping
Schema-log mapping
Generate traces
Event log
Artifact lifecycle discovery
Apply general process discovery techniques
Lifecycle model

Figure 10: Overall method

Schema extraction may be required to create a structured dataset with a known schema (which is the prerequisite to find artifact model(s)).

The dataset is then used to identify artifacts and the corresponding database schema for each artifact – this is called the ***artifact schema***. Each artifact schema is assumed to be a subset of the complete schema.

For a specific artifact schema (and the related part of the dataset) it can then be determined what event types exist and how these are stored in the dataset. This information can be used to create a mapping from the artifact schema to events.

The mapping and the structured dataset can subsequently be used to generate traces suitable for process mining. Finally, these traces can be used as input for any existing process discovery technique to generate the lifecycle for each artifact.

2.1 Schema extraction

Schema extraction is defined here as extracting structural information (e.g. candidate keys and relationships between entities) from structured data (e.g. tables). It takes as input a set of structured (tabular) data and produces (1) a primary keys for each table, (2) the domain for each element (column) in each table and (3) the relationships (foreign keys) between tables.

Schema extraction is often seen as a technique within the information extraction domain (i.e. extracting structure from unstructured data [32]) [33], [34], but considering the context of this method it seems to make more sense to define it as a separate step. This has to do with the nature of ERP systems: most ERP systems already contain structured data (in the form of a relational database), but schema information such as keys and relations is often not available [16], [17]. ERP systems are the focus of the this method, thus it makes sense to define schema extraction as a separate step.

Several techniques exist to extract schema information from a structured dataset [35–38]. Of specific interest is the domain for each column: since events are ordered by occurance[2] we are looking for ways to identify this order in the data. If a domain is ordered by occurance (i.e. sequential or timed) then the column can be used to discover the order of the events. Therefore this information should also be recorded in this step. An approach to execute schema extraction is described in section 3.2.

Formally the input of this step is a set of tables $\mathcal{T} = \{T_1,...,T_n\}$, with each $T_i = (\mathbf{C}, \mathbf{C_p})$ a table that contains a set of columns $\mathbf{C} = \{C_1,...,C_n\}$ and an unknown primary key $\mathbf{C_p} = \{\}$. Both $\mathbf{C}$ and $\mathbf{C_p}$ are a subset of the set of all columns $\mathcal{C} = \{C_1,...,C_n\}$. For each column C_i the domain D is unknown.

The output is a schema $\mathfrak{S} = (\mathcal{T}, \mathcal{F}, \mathfrak{D}, \textit{column\_domain})$ with $\mathcal{T}$ a set of tables (as above) and $\mathcal{F} = \{F_1,...,F_n\}$ a set of foreign keys. For each table the primary key $\mathbf{C_p} = \{C_{p1},...,C_{pn}\}$ is known. Each $F_i = (T_p,\mathbf{C_p},T_c,\mathbf{C_c})$ is a foreign key from parent table T_p with primary key C_p to child table T_c with referencing columns C_c. Finally the complete set of domains $\mathfrak{D} = \{D_1,…,D_2\}$ is identified and there is an assignment function *column_domain* : $\mathcal{C} \rightarrow \mathfrak{D}$ that assigns a domain to each column.

2.2 Identify artifact schemas

In this step the subset of the dataset that is of interest for each specific artifact is identified. The idea is to partition the full schema in a number of clusters (one for each artifact) and assign each table to one or more clusters. In addition a representative main table will be chosen for each cluster. This ***main table*** contains the instance information for the specific artifact: each artifact instance can be identified by the primary key of the main table. Of course the underlying assumption is that such a table exists for each artifact, but my own experience and previous research [15], [17], [21], [22] suggests that this is usually the case.

Previous work on database schema and graph summarization showed a few key points should be taken into account while identifying artifact schemas:

- Important tables should not be contained in the same cluster. As explained in [39] important is a subjective term, but there are some objective measures that indicate importance of a table. Specifically for artifact clustering the tables that can be used to identify artifact instances should be regarded as highly important

[2] Events that occured first are ordered before events that occured later.

and must be contained in separate clusters. Aside from this, high connnectedness (a large number of incoming and outgoing foreign keys) and high data cardinality (large numbers of different data values) are indicators of higher importance of a table [39].
- Clusters should be evenly distributed over the schema. This corresponds to the fact that all information about an artifact should be contained in its schema. Intuïtively a schema containing 40 tables is unlikely to be represented correctly by 3 clusters with 2 of them each containing a single table and 1 cluster containing the other 38 tables, especially if the two single table clusters are directly connected via a foreign key. If one focusses only on importance this can be the result, however. One measure for this is the density of the clusters: the number of links (foreign keys) between tables as a ratio versus the number of possible links between tables. It is to be expected that the intra-cluster density is higher then the average or inter-cluster density [40]. Another related measure is the coverage as defined in [39]: this measure is calculated using the cardinality of the foreign keys and the foreign key path length between each table and the main table.
- The number of clusters created should be as small as possible [39], while maintaining sufficient information about the schema as a whole. For artifact schema clustering this means that the granularity of the clustering should match the expected granularity of the artifacts: e.g. if order invoices and orders are expected to be different artifacts they should be shown as different clusters.

These three points are related to each other and tradeoffs will need to be made for each set of clusters that is calculated. The exact importance of each depends on the required granularity of the summary and the specific dataset. An approach to execute artifact schema identification is described in section 3.3.

Formally the input of this step is a schema $\mathcal{S}$ as defined above. The output is a set of artifact schema's $\{\mathcal{S}_{A1},..., \mathcal{S}_{An}\}$ where each artifact schema $\mathcal{S}_A = (\mathcal{T}_A, \mathcal{F}_A, \mathcal{D}_A,$ *column_domain*, $T_m)$ is defined as a schema with a main table $T_m \in \mathcal{T}_A$ and the other attributes defined as done for any schema above.

2.3 Create schema-to-log mapping

Creation of the mapping between the artifact schema and a resulting event log is done in this step. It takes as input (1) an artifact schema and (2) a structured dataset described by the artifact schema and produces (1) a set of event types identified in the dataset and (2) a mapping from the dataset to these event types. The mapping found in this way describes how to extract the different events from the dataset. Note that in this case the purpose of the event log is to use it to discover the lifecycle of an artifact using process mining techniques. Since it depends on the goal of a process mining project what to include in an event log [9], [41] this should be taken into account.

All known previous work on support for this step assumes that the mapping is created manually by domain experts [15], [21], [41], [42]. In addition to this a variety of papers are available that describe how event logs were extracted from a specific system [15–17], [23], [43–45]. The artifact-centric approach already solves some challenges given in these papers. Therefore these no longer have to be taken into account:
- Convergence and divergence [15], [16], [41]. As explained in the introduction the artifact-centric approach was developed to handle problems with convergence and divergence. Thus these problems should not play a role when working with a single artifact.
- Traces should contain only events that belong to a single process [41]. For the artifact centric approach this implies that all events should be related to a single artifact. Since the previous step ensures that all input data (including events) is related to a single artifact, this should not play a role here.

An overview of more general problems for event log extraction is given in section 4.2, followed by an approach to identify artifact-to-log mappings.

Formally the input of this step is an artifact schema $\mathbb{S}_A$ as defined above. The output of the step is an artifact specific log mapping LM as defined in Table 2 below. Figure 11 shows the same information as a class diagram. This mapping describes the conversion of a dataset to an event log in the eXtensible Event Stream (XES) format. Note that a trace mapping TM defines the conversion to any general event log, while the XES specific concepts (e.g. classifiers) are part of the log mapping LM.

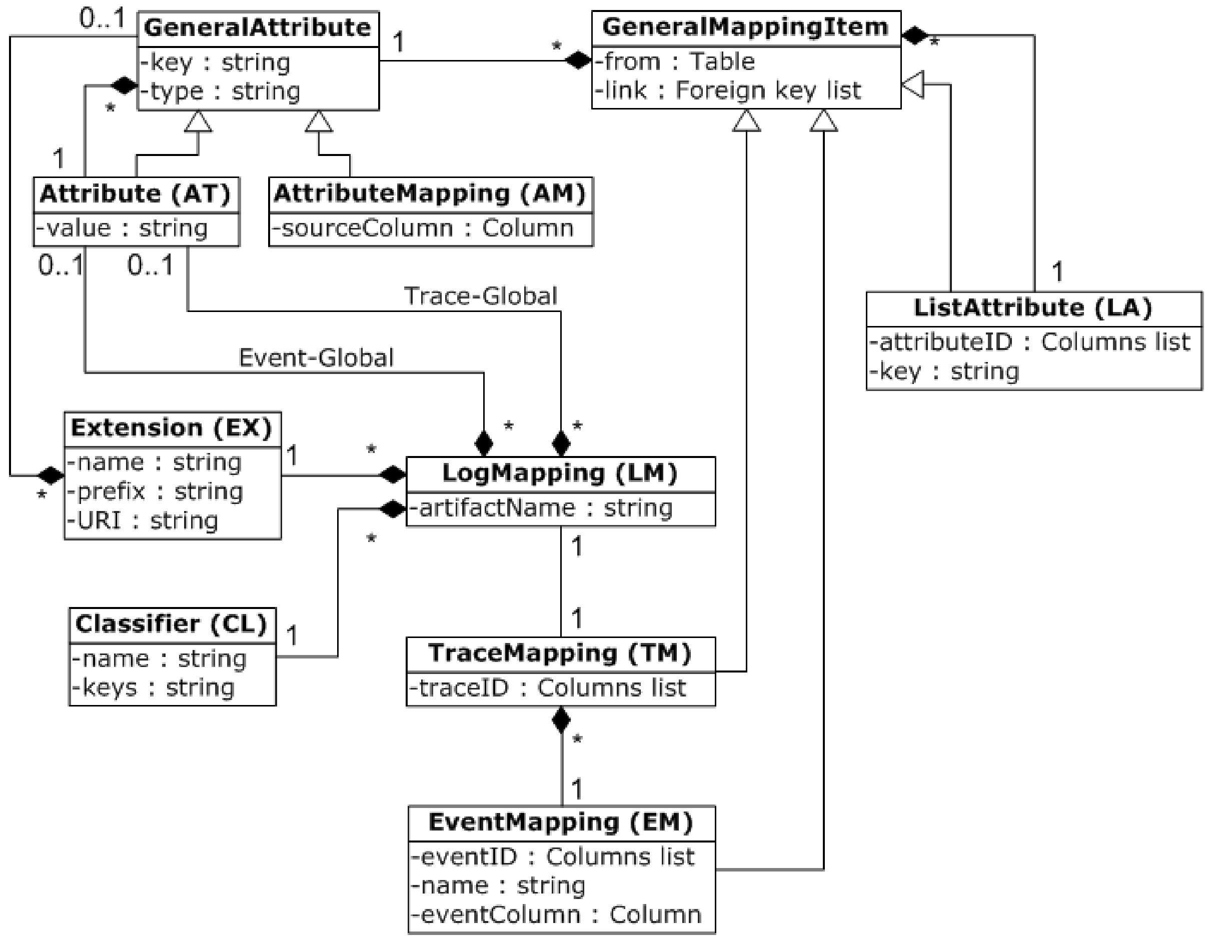


Figure 11: Class diagram of mapping domain model

The model is similar to that of XESame [41], except for a ListAttribute that is used to describe a variable length list of attributes. This was added because XESame cannot handle these. Aside from this the mapping described here can easily be translated to the XESame model to allow for manual adjustments to the identified mapping. See Appendix C for a description on how to do this.

Symbol	*Description*
LM = (name, TM, **EX**, **CL,** $\mathbf{AG_T}$, $\mathbf{AG_E}$)	Log mapping with artifact name *name*, a trace mapping TM, extensions **EX**, classifiers **CL**, global trace attributes $\mathbf{AG_T}$ and global event attributes $\mathbf{AG_E}$
TM = ($\mathbf{C}_{\mathrm{TID}}$, T_{From}, $\mathbf{F_{Link}}$, **EM**, $\mathbf{AM_T}$, $\mathbf{LA_T}$)	Trace mapping with traceID columns $\mathbf{C}_{\mathrm{TID}}$, main table T_{From}, other table links $\mathbf{F_{Link}}$, event mappings **EM**, attribute mappings $\mathbf{AM_T}$ and list attributes $\mathbf{LA_T}$
EM = {EM_1,...,EM_n}	Set of event mappings

Symbol	*Description*
EM = (name, $\mathbf{C}_{EID}$, C_e, T_{From}, $\mathbf{F}_{\mathbf{Link}}$, $\mathbf{AM}_{\mathbf{E}}$, $\mathbf{LA}_{\mathbf{E}}$)	Event mapping for event *name* with eventID columns $\mathbf{C}_{EID}$ and event column C_e, main table T_{From}, other table links $\mathbf{F}_{\mathbf{Link}}$, attribute mappings $\mathbf{AM}_{\mathbf{E}}$ and list attributes $\mathbf{LA}_{\mathbf{E}}$. The event column describes the ordering of the events, most likely containing timestamp values
LA = {LA_1,...,LA_n}	Set of n list attributes
LA = (key, $\mathbf{C}_{AID}$, T_{From}, $\mathbf{F}_{\mathbf{Link}}$, $\mathbf{AM}_{\mathbf{L}}$, $\mathbf{LA}_{\mathbf{L}}$)	List attribute (an attribute with multiple values) mapping with given *key*, attributeID columns $\mathbf{C}_{AID}$, main table T_{From}, other table links $\mathbf{F}_{\mathbf{Link}}$, attribute mappings $\mathbf{AM}_{\mathbf{L}}$ and list attributes $\mathbf{LA}_{\mathbf{L}}$
AM = {AM_1,...,AM_n}	Set of n attribute mappings
AM = (key, type, C_a)	Attribute mapping with given *key*, *type* and attribute column C_a
AG = {AT_{G1},...,AT_{Gn}}	Set of n global attributes
AT = (key, type, value)	Attribute with given *key*, *type* and *value*
EX = {EX_1,...,EX_n}	Set of n extensions
EX = (name, prefix, URI)	Extension with given *name*, *prefix* and *URI*
CL = {CL_1,...,CL_n}	Set of n classifiers
CL = (name, keys)	Classifier with given *name* and *keys*

Table 2: Log mapping symbols

2.4 Generating traces

Using the mapping found in the previous step events can be generated from the dataset. The approach described in [41] was designed for a similar purpose: it takes as input a dataset and a mapping and produces an event log. Similarly for each artifact this step should take as input the dataset and the mapping for the artifact as decribed in the previous section. The output should be an event log in the eXtensible Event Stream (XES) format; The XES format was chosen since this is the only available standardized event log format. It was designed for the interchange of event log data in a simple, expressive and flexible format, while allowing for extensions of the format in a transparant manner [46]. An approach to generating traces given a mapping and a dataset is described in section 4.3.

Formally this step takes as input the dataset and the mapping LM as decribed in the previous section. The output should be a XES log file as defined by [46]. The XES format defines an event log as a set of traces each containing a list of events. All of these can have attributes that provide more information; there are 5 different types of attributes. Extensions can be defined that give more meaning to an attribute: each extension defines a list of attribute keys with a specific meaning. Global attributes can be defined for both traces and events: these are attributes that guaranteed to be available for all traces and events respectively. Finally event classifiers provide a way to compare events. Each classifier specifies a number of event attributes which can be used together to uniquely identify all events.

2.5 Apply process discovery techniques

The goal of ***artifact lifecycle discovery*** is to discover both the internal lifecycle of an artifact and its interaction with other artifacts, thereby fully describing how an artifact operates. With the event log produced in the previous step a variety of process discovery techniques can be used to generate the lifecycle for each artifact. Section 4.4 describes a number of these existing techniques and how to apply them to discover the internal lifecycle of an artifact.

Chapter 3 – Artifact schema identification

The database of an ERP system contains the data of the entire system, but it may not be intuïtively clear how this data is structured. A way to describe the structure of the data is by identifying all artifact schema's that are part of the database. An artifact schema is a relational schema describing the data of an artifact; it is a way to describe the information model of the artifact. Note that if all artifact schema's are to be identified this implicitly includes identifying all artifacts present in the system.

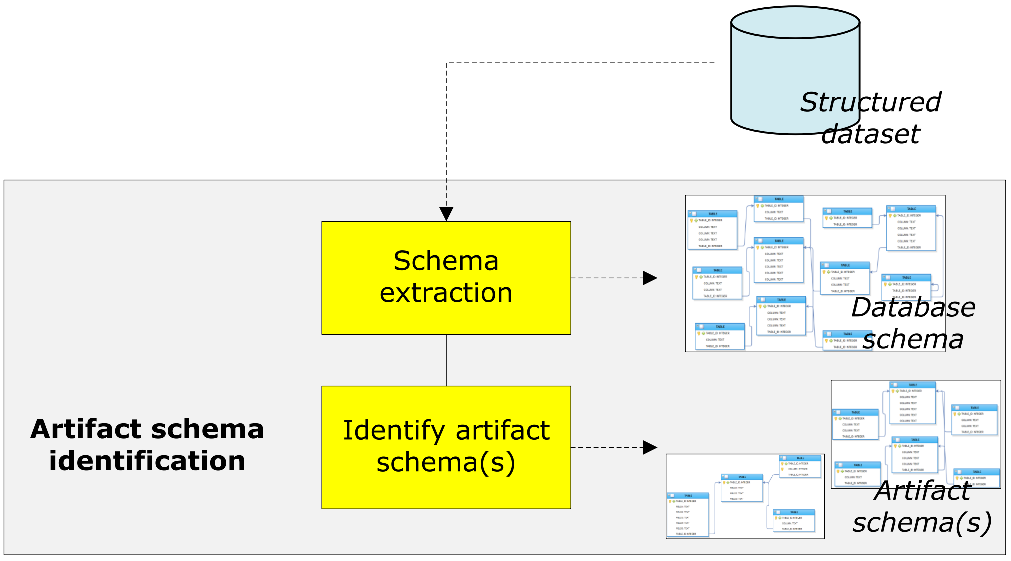


Figure 12: Artifact schema identification approach

In this chapter a two-step approach is presented to identify all artifact schema's that are part of an ERP systems database. First a variety of techniques are presented to rediscover several types of metadata (column domains, primary keys and foreign keys), since this metadata is commonly missing in ERP system databases. The result of this is a complete schema for the entire database. Then a fuzzy clustering approach is presented that selects subsets of the entire database schema: the artifact schemas. Both steps take into account that the lifecycle of each artifact will need to be identified afterwards.

Since the focus of this report is on ERP systems that rely on relational databases the terminology that is relevant in that context is first introduced. Then the steps to discover the complete schema and identify the artifact schemas are presented. Each of the steps consists of an overview of related previous work followed by an approach to execute the step.

3.1 Relational databases

A relational database is a shared collection of logically related data described by a relational model [30], [31]. The relation model describing the database is called a *schema*. This schema contains both the tables and the relations between those tables.

Each ***table*** consists of one or more ***columns***, each having a name and a ***domain*** (the set of possible values that can occur for that column). A row of data (one field for each column) is called a ***record***.

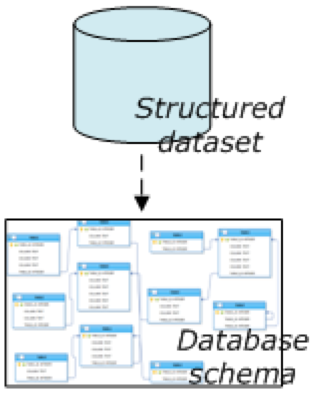
A ***functional dependency*** occurs when the values in one set of columns can be used to determine the value of another column. Formally a functional dependency is noted as **C** → **C'** with both **C** and **C'** a set of columns. The values of **C'** are functionally determined by **C**. In this case it is said that the columns in **C** functionally determine **C'**.

A table may also have one or more ***candidate keys***: a set of columns that can be used to uniquely identify all records in the table. A candidate key is a specific type of functional dependency: the candidate key can be used to determine the value of all other columns in the same table. A candidate key must be minimal: it should not be possible to uniquely identify all records using a subset of columns of the candidate key. One of these candidate keys is often chosen to be the ***primary key*** for the table.

An ***inclusion dependency*** (IND) occurs when all values in one set of columns are included in all values of another set of columns. Formally an inclusion dependency is noted as $(C_1,...,C_n) \subseteq (C'_1,...,C'_n)$, with both $(C_1,...,C_n)$ and $(C'_1,...,C'_n)$ a sequence of columns. The values of $(C_1,...,C_n)$ are pairwise included in $(C'_1,...,C'_n)$: the values of the child column combination $(C_1,...,C_n)$ are a subset of the values of the referenced column combination $(C'_1,...,C'_n)$. In this case it is said that the child columns are included in the referenced columns.

Relations between tables are described by ***foreign keys*** between parent and child tables. For a foreign key a set of columns is chosen in the child table. These columns reference the primary key of the parent table: The values of the child column combination are a subset of the values of the referenced primary key. Thus a foreign key is a specific example of an inclusion dependency. A foreign key can be a 1-on-1 reference or a 1-on-n reference: a 1-on-1 reference means that each primary key value of the parent occurs at most once in the child table, while for a 1-on-n reference each primary key value may occur multiple times in the child table.

3.2 Schema extraction



As noted before schema extraction is extracting structural information (e.g. candidate keys and relationships between entities) from structured data (e.g. tables). It takes as input a set of structured (tabular) data and produces a relation schema for the data. This schema contains (1) the domain for each column in each table, (2) a primary keys for each table and (3) the foreign keys between tables. Extracting each of these types of information is a distinct subproblem that has been analyzed in the past. Therefore each subproblem (domain extraction, primary key extraction and foreign key extraction) will be treated separately in this section.

3.2.1 Domain extraction

Overview

In the context of artifact discovery ***domain extraction*** is the grouping of columns into a number of domains based on the data values and known meta data (e.g. name, data type) of the columns such that these domains can be used for schema-to-log mapping. This means that it must be possible to map each domain to exactly one XES datatype and to specify for each domain if the data elements are ordered by occurance. Since we assume that primary and foreign key information is not available yet at this step, this information cannot be used.

The simplest approach for domain extraction is to let a domain expert look at all the column combinations to classify them as "the same" and "not the same". This would create groups of columns that are "the same", meaning that they share the same domain. After this is done the domain expert should classify every domain as ordered by occurance or not. In practice manually comparing all column combinations is infeasible due to the large number of possible column combinations. Therefore previous research has suggested to use a variety of distance measures as an alternative to manual classification; such a distance measure represents the likelyhood that columns are similar. The interpretation of these distance measures is done by so-called clustering algorithms that result in groups of columns that are similar according to the distance measure.

Related work: Domain extraction

In general ***domain extraction*** is the grouping of columns into a number of attribute types based on the data values and known meta data of the columns. The idea is that all columns with the same domain should be assigned to the same attribute type; the attribute type identifies the domain. E.g. the *cd_name* column of various tables should be grouped together to form the cd name attribute type, while the *customer_name* columns should not be in the same group (even though they have the same data type).

Only a limited amount of work has been done on domain extraction. The authors of [37] experimentally compare a variety of methods to cluster columns together based on the so-called q-gram signatures of the data in the columns while in the work presented in [47] clusters of columns are computed using the distribution of the data in the columns. In addition to this the slightly more general problem of grouping columns is shown to be NP-Complete in [48].

All methods compared in [37] use a distance metric based on the q-gram signature of each column as a basis. It is noted that accuracy is always highest when cosine distances are used. Q-gram signatures given by principal component analysis (PCA) on the column x column covariance matrix are shown to have the highest accuracy and to scale reasonably well. Another option with a high accuracy over all experiments is information bottleneck (IB) based clustering, although this is slightly slower than the PCA based method. In addition it is shown that min-hash signatures are the fastest option, giving a reasonable accuracy as a result.

The two-step approach described in [47] starts with a rough clustering using the distance between the combined value distribution of each column pair as a basis. The clusters identified in this way are then further refined using the distance between the intersection value distribution of each column pair as a basis. For the seconds step the authors also introduce witness columns: the assumption is that if column C_A is related to a witness column and column C_B is related to the same witness column that then it is highly likely that C_A and C_B are also related. In addition to the basic approach the authors also introduce a variety of performance optimizations.

Earlier work on column similarities are that on Bellman [49] and the application of information bottleneck for general column clustering [48]. The work on Bellman suggests the use of q-gram sketches and q-gram signature to verify column similarities, but does not mention how columns should be grouped together. The approach described in [48] shows how columns can be clustered, but not with the purpose of domain based categorization in mind.

Finally, as mentioned before the information bottleneck method introduced in [50], [51] can be used to cluster columns together. Since the introduction of the method two more efficient versions of the algorithm were introduced, as described in [52] and [53].

Approach

A two-step approach is suggested for domain extraction in this context:

1. Heuristically determine the XES datatype of each column by looking at its technical datatype or data values.
2. Apply general domain extraction techniques such as those described in [37] and [47] to each set of columns sharing the same XES datatype.

Aside from determining the XES datatype the first step also makes sure that each column containing time values is identified, thereby likely specifying most or all columns that can be used to identify events. In addition to this the heuristics provide a way to reduce the size of each set of columns to be clustered by looking at each column individually. Since the second step is NP-complete this may reduce the total computation time significantly.

Heuristically determining the XES datatype is fairly straightforward. If the technical datatypes of all columns are trustworthy, then determining the XES datatype is trivial. If the technical datatypes cannot be relied upon then the values of each column can be checked using generally accepted heuristics. A column can for example only be of the boolean type if it contains only "true", "false", "0" or "1" values. The other XES datatypes can be determined using similar heuristics.

For the second step a general domain extraction technique can be used. Based on the evaluation in [37] a q-gram signature method using principle component analysis should provide good results. For large datasets min-hash functions appear to be the best alternative. The authors do not provide a suggested clustering method in their paper to use with calculated distances. Since the number of domains is not known beforehand this should be a method that calculates the number of clusters as part of the algorithm, such as DBScan [54][3].

Sampling can be used to improve the efficiency each technique by reducing the number of values that need to be checked for each column. For both the heuristics given above and functional dependency extraction the results using sampling are correct if there are no "invalid" records in the sample while these exist in the complete dataset. Thus formula (1) can be used to approximate the required sample size, given an allowed fraction of invalid records in the complete dataset ε and a maximum probability that an invalid record is missed δ [55]. For a more exact (but computationally harder) estimation the method of [56] can be used.

$$\text{Sample size} \approx \frac{(\#\mathrm{T})^{1/2}}{\varepsilon} \log \frac{1}{\delta} \qquad \textbf{(1)}$$

3.2.2 Primary key extraction

Overview

Primary key extraction is the identification of the primary key for a table (for which no such key is defined) based on the data values and known meta data of the table. Typical metadata that can be used includes the column names, positions or datatype. It involves both the extraction of candidate keys and the selection of the primary key from these candidate keys. The extraction of candidate keys can be defined exactly and thus this can in theory be fully automated. This is not the case for the selection of the primary key, since this is subject to assumptions known to the persons responsible.

The simple approach to primary key extraction is to first check each column combination to see if it is a candidate key and then let a domain expert pick one of these keys as the primary key. Checking if a column combination $\mathbf{C_X}$ is a candidate key can be done by grouping all records in the table based on the values of $\mathbf{C_X}$. For each value combination only one record should exist. Thus if a group exists that contains more than record then the column combination is not a candidate key. The disadvantage of this approach is that all records in the table need to be grouped together for each possible column combination. Although the grouping of records can be done efficiently by sorting, the number of column combinations increases exponentially with the number of columns. As a result this approach is not feasible for even a relatively small table. Because of this a number of more efficient approaches have been suggested in previous research.

[3] Although the value distribution approach described in [47] and an information bottleneck based approach with one of the more recent faster algorithms [52], [53] would seem promising these were not evaluated due to time constraints.

Related work: Candidate key extraction

Candidate key extraction is the identification of candidate keys for a table (for which no such key is defined) based on the data values and known meta data of the table. A significant amount of work has been done on this subject and the related subject of functional dependency extraction. In addition to data-driven approaches a variety of approaches have been described that infere functional dependencies or candidate keys from the queries that were executed against the database. These approaches are not discussed here since it is assumed that information about queries executed is not available in this case.

Two main types of algorithms exist to extract candidate keys or functional dependencies from the data of a table: level-wise or a-priori algorithms (all based on [57]) and record-based algorithms. Level-wise algorithms check each column-combination once, while record-based algorithms check each record once. Since discovering a unique column combination (i.e. candidate key) of a table is an NP-Complete problem [58], all of the algorithms can take an excessive amount of time to complete in the worst case scenario.

The only record-based approach that I am aware of is Gordian [59]. The idea of Gordian is to eliminate non-keys instead of discovering candidate keys directly. Experiments on various datasets show that Gordian scales approximately linearly with the number of columns and records in practice. In [60] Gordian is shown to perform poorly when large numbers of non-uniques are found however, since then unique generation takes long. To improve on this the author proposes a hybrid approach called HCA-Gordian [60], [61] to identify candidate keys. HCA is a level-wise approach that includes various optimizations of previous level-wise approaches such as TANE [62], Bellman [49], FUN [63] and the pruning mechanism based on Armstrong's axioms described in [64]. The authors argue that level-wise algorithms tend to take long when large numbers of uniques are found, which is the exact opposite situation of the worst-case scenario for Gordian. Their hybrid approach uses Gordian to quickly prune large numbers of non-keys using a sample of the data and then HCA to determine keys. They experimentally show that HCA-Gordian outperforms various level-wise approaches and Gordian in most cases, while not performing much worse in cases where the level-wise or Gordian approach was optimal.

Aside from the exact approaches given above, some work has been done on approximate identification of functional dependencies. The basic idea is that approximate results using only a sample of the data can be obtained faster than exact results for which all of the data is required. The CORDS approach [65] uses chi-squared analysis on a sample of the data for this purpose. A limitation of the approach is that only single column functional dependencies are identified. The authors show that due to statistical properties a sample of a few thousands rows should provide acceptable results regardless of the database size. Kivinen et al. gives a more exact result in [55]. They define exactly what fraction of records needs to be sampled to guarantee that a functional dependency is valid with a given confidence.

Approach

Although existing methods are available for the extraction of candidate keys, this is not the case for the selection of the primary key. Therefore an existing method will be used for candidate key extraction, while some heuristics are defined for the selection of the primary key.

The most promising approach for candidate key extraction appears to be HCA-Gordian [60], which discovers all candidate keys of a table. Since only the primary key is required a slight modification to this approach is made though. Both the Gordian and HCA step are run with a small sample of the data. This results in a number of possible candidate keys, including a number of false positives. These possible candidate keys are then ordered using the heuristics described below and verified against the complete dataset. The first *n* candidate keys that are verified to be correct are the most likely primary keys. Thus after *n* candidate keys are verified to be correct the algorithm can terminate.

Although no work appears to exist on the selection of a primary key from a set of candidate keys, practical experience suggests that some guidelines are usually followed when a database is designed:

1. A small key is preferred to a larger key. Thus the number of columns in a primary key should be as small as possible.
2. Key columns tend to be positioned at the start of a table. Thus the total position numbers of a key column combination should be as small as possible.

These guidelines can be used to order a set of candidate keys by descending probability that they are the primary key of the table. First columns should be ordered on the number of columns they contain. If this is equal they should be ordered by the total position number of the columns in the combination. If that is equal as well they should be ordered lexicographically by comparing the position of column pairs 1 through ℓ of each combination, with ℓ being the number of columns in each combination.

3.2.3 Foreign key extraction

Overview

Foreign key extraction is the identification of candidate foreign keys between a pair of tables based on the data values and known meta data of the table. Typical metadata that can be used includes the primary keys, column names, positions or domains. Similar to the two steps of primary key extraction it involves both the extraction of inclusion dependencies (IND's) and the selection of actual foreign keys from these IND's. Also in this case the first step can in theory be fully automated, while domain knowledge is required to select the actual foreign keys.

The simplest approach to identify a foreign key between a pair of tables is to first check if inclusion dependencies exist between the tables and then let a domain expert pick the actual foreign keys from these inclusion dependencies. Checking if an inclusion dependency exists can be done by comparing all pairs of column combinations with the same number of columns. For each pair one would check if all the values in the column combination of one table also exist in the column combination in the other table. If this is the case then the column combination pair is an inclusion dependency. The disadvantage of this approach is that all pairs of column combinations need to be compared. Again the number of combinations increases exponentially with the number of columns, making this approach infeasible for even a relatively small table. Because of this a number of more efficient approaches have been suggested in previous research. In addition to this the resulting number of IND's can still be large, making it hard for a domain expert to pick the actual foreign keys. Therefore a number of measures have been suggested that represent the likelyhood that an IND is actually a foreign key.

Related work: Foreign key extraction

A significant amount of work has been done on the subject of foreign key extraction and the related subject of inclusion dependency (IND) extraction. According to [66] deciding if there exists a multi-column inclusion dependency (between two tables) is an NP-Complete problem. They expect results to be better in practice though, since the NP-Hardness was shown on a highly artificial dataset.

Two similar approaches for IND extraction are MIND [67], [68] and SPIDER [69], [70]. Both algorithms first verify all single column IND's in parallel, thus requiring only a constant number of passes over all data (regardless of the number of IND's to verify). For this both algorithms preprocess the column data prior to verification. A level-wise approach is then used for higher order IND's in which IND candidates for each level are generated from previously verified IND's using a set of inference rules specified in [67]. There are two notable differences between the algorithms:

- MIND creates an inverted index of the column data to verify single column IND's, while SPIDER prepares the data by creating a sorted list of values for each column.
- IND's of level 2 or higher are verified sequentially against the database with MIND, while SPIDER verifies these similarly to single column IND's (i.e. in parallel).

For both algorithms it is also shown how approximate IND's can be extracted if inconsistencies exist in the data [68], [70]. An attempt was done to use SPIDER on a SAP dataset with ¼ million columns, but this could not be tested due to limitations on the number of open files (the algorithm creates 1 tempory file per column) and main memory constraints of the JVM. For both MIND and SPIDER it is shown that the runtime increases linearly with the data size for IND's consisting of a limited number of columns.

When the number of columns on both sides of an IND exceeds 8 to 10, level-wise approaches such as MIND and SPIDER start to exhibit exponential running time increases [67], [71–73]. Two approaches to find larger IND's are FIND$_2$ [72] and Zigzag [71]. Both algorithms are based on a property of IND's that allows smaller IND's to be deduced from larger IND's. The algorithms use jumps in the search space: given a set of verified 1 and 2 column IND's large IND candidates are constructed and verified before deducing smaller IND's from these verified large IND's. Even though scalability of these algorithms is shown to be better than those of level-wise algorithms if the number of columns exceeds 10 [72], [73], the practical relevance of them may be limited since real databases usually do not contain IND's with more than 6 columns [70], [72].

A number of approaches exist that focus on identifying true foreign keys from IND's that were shown to hold in the data. Both [73] and [35] focus only on the data for this purpose, while [74] also includes simple meta data properties (such as column names). In [73] it is shown that theoretically the probability that an IND is valid by chance is greater than 5% if the included column combination contains less than 7 distinct values. Another good predictor if an IND is a foreign key is the value distribution of both included and referenced column combinations; These should be similar for true foreign keys [35], [73]. While in [73] the χ^2 test for independence is used to prune invalid IND's, the Earth's Movers Distance (EMD) is shown to be a good predictor of the validity of a foreign key that can be calculated very efficiently in [35]. The authors note that their experiments always show one or more cut-off points for the calculated EMD's; one of these cut-off points always is the boundary between true and false foreign keys. The authors of [73] note that the value distribution can result in false negatives if the included column combination is a subset within a specific range of the values of the referenced column combination. Therefore they propose that a value distribution test should only be used if the number of distinct values in the included column is less than 7.

The authors of [74] use a slightly different approach: they specify a number of properties that could indicate valid foreign keys and then use existing classification algorithms to separate true foreign keys from IND's that are coincidentally correct in the data. Thus their approach requires a training dataset of valid and invalid foreign keys. Experimental evaluation shows that the approach has difficulties handling empty tables, transitive foreign keys, small tables and one-to-one foreign keys [74]. In addition [35] shows that the data based properties are all captured well by the value distribution property as discussed above.

Approach

To extract foreign keys an integrated approach can be constructed from both existing methods for the extraction of IND's and the selection of foreign keys from IND's. From a mathematical point of view the IND extraction problem is slightly harder than primary key extraction: candidate key extraction is NP-hard when considering a single table, while IND extraction is NP-hard when considering a pair of tables. Since there are $|\mathcal{T}|$ tables and $|\mathcal{T}|^2$ combinations of tables in the schema this implies that $|\mathcal{T}|$ NP-complete problems must be solved for primary key extraction, while $|\mathcal{T}|^2$ NP-complete problems must be solved for foreign key extraction. Thus this problem can only be tackled by an efficient approach.

The base approach chosen for IND extraction is that of SPIDER [69], [70], although the preprocessing of the data is done using an inverted index (as suggested by [67], [68]). The creation of the inverted index is done using single-pass-in-memory indexing as described in [75]. This reduces the number of sorts required when compared to SPIDER, since values only need to be sorted when memory is exhausted and the dictionary is written to a file. Similarly this reduces the number of required files. In two cases the original SPIDER algorithm is likely to perform better:

- If a table contains such a large number of distinct values that the values of a single column combination do not fit into memory then the number of sorts and the number of files created actually increases. Note that the sorting will still need to be done by the database management system (DBMS), which may in practice suffer from the same limitations.
- The existence of indices for some column combinations: in that case the values for these column combinations are already sorted in the DBMS and thus returning the column combination in sorted order saves one sort.

To reduce the number of IND candidates to be tested two heuristics are used:

- The domains of each column pair must match. So a column with the "string" domain is not tested to be included in any column with an "integer" domain.
- The referenced side of a candidate IND must be a primary key. Thus no column combinations were tested to be included in a column combination that was not a primary key.

An intented effect of the second heuristic is that a large number of 1-column IND's are not tested. Since these are required for the pruning mechnism described in [67], [68] this mechanism cannot be used[4].

The approaches described in [71], [72] are not chosen since they were only shown to be beneficial if the number of columns on each side of the IND exceeds 8, while this number rarely exceeds 6 in practice [70], [72].

A variety of properties are calculated that allow for the selection of the actual foreign keys from the identified IND's. The accuracy of each of these properties is evaluated empirically in subsection 0.

1. The number of distinct values of the included column combination.
2. The earth movers distance (EMD) between the value distribution of the referenced and included column combinations. The EMD is used instead of the χ^2 test since it appears to be possible to test it more efficiently. The exact measure used is the thresholded EMD as decribed in [76], since it was shown to be a better metric for the difference between two value distributions than the general EMD.
3. The number of possible referenced column combinations of an included column combination.
4. Typical name suffixes of the included column combination. This is calculated as the number of columns for which the name ends with "ID", "NR" or "NO" divided by the total number of columns in the combination.

[4] Due to time constraints it was not tested if the overal impact on efficiency was positive or negative as a result.

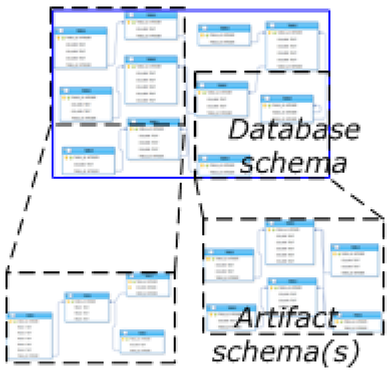
5. The longest common subsequence (LCS) [77] that is shared between the names of the referenced and the included column. For this the name is taken to be the combination of the table name and the column name (e.g. for column "reqid" in table "quote" the used name was "quote_reqid"). The ratio is calculated as the length of the LCS divided by the length of the shortest name. The score for the IND is the average ratio between each column pair.
6. The Dice coefficient [78] between the names of the referenced and the included column. For this the name is also taken to be the combination of the table name and the column name. The Dice coefficient is defined as the number of shared character bigrams as a ratio of the total number of bigrams in both words. The score for the IND is the average coefficient between each column pair.

The first 4 properties were all defined in previous research. The last 2 are defined since [74] suggests that more sophisticated ways to compare names should be investigated. The LCS ratio is expected to be a good predictor of valid foreign keys, since included columns often have a name of the form "[referenced table name]_[referenced column name]". The Dice coefficient is included for comparison with the LCS ratio, since it is generally considered to be a good similarity measure for words [78].

3.3 Identify artifact schemas

Overview

Artifact schema identification is the selection of the tables, foreign key relations and main table from a database schema for each artifact based on the fully specified structural information and data values of the database. Notable here is that it is highly likely that some tables are assigned to multiple artifact schema's. An example would be the *cd_quote_order* table in the CD shops database, that is part of the schema of all three artifacts.



Similar to domain extraction the simpest approach is to let a domain expert first pick all tables that represent the artifact instances and then let this expert assign the remaining tables to one or more of these "artifact instance tables". Since there can be thousands of tables in a database, often with cryptic names, this is not feasible in practice. Therefore some way to measure the likelyhood that a table contains artifact instances needs to be developed. In addition to this a distance measure between tables can be used to represent the likelyhood that a table needs to be assigned to a table containing artifact instances. As with domain extraction the interpretation of these distance measures can be done by so-called clustering algorithms that result in groups of tables that are close according to the distance measure.

Related work: Schema summarization

Schema summarization is the (automatic) creation of a summary of a database using the structure of and/or data contained in the database . A schema summary consists of a set of abstract elements and links that is representative for the underlying database. The identification of artifacts and their schema's can be seen as a specific form of schema summarization. Although manual schema summarization has been studied for over two decades [79], only recently a few papers where published on the topic of automatic schema summarization. All of these papers build on the more widely studied field of graph-based clustering techniques; for an extensive overview of these techniques the reader is refered to [40].

The first work on automatic schema summarization using both data and structure information is [39]. The authors define three properties of summaries that can be used to measure how good a summary is: *complexity* (the number of elements in the summary), *importance* (of elements contained in the summary) and *coverage* (how representative the elements in the summary are for all elements in the schema). In addition to this the *BalancedSummary* algorithm is described that balances the importance and coverage properties of a summary given a required summary size. The authors show experimentally that summaries created by the algorithm are as good as summaries created by human experts.

Since entropy is considered to be the standard metric for information content of data [80] a number of papers recommends its use for schema summarization [32], [36], [48], [81], [82]. ***Entropy*** is a measure of uncertainty of a random variable X. It is defined by $H(X) = -\sum_{x \in X} p_X(x) \log p_X(x)$. When applied to data values it takes on its maximum value when each value is unique. The authors of [48] define how this might be used for information bottleneck based clustering of tables, but do not show any method or result with their metric. Yang et al provides an approach that first calculates the importance of a table using entropy combined with structural (foreign key) information, then calculates distances using data information and finally calculates clusters with the previously calculated numbers [32], [36], [81].
The distance between two tables connected via a foreign key is defined using two properties:

1. The distance between two tables is inverse proportional to the fraction of records in the parent for which a child record exists (the ***matched fraction***).
2. The distance between two tables is proportional to the average number of records in the child for each parent record for which a child record exists (the ***matched average fanout***).

The summary is then calculated using a weighted k-center algorithm: this algorithm takes the importance of tables into account when determining the distance, making it more likely that highly important tables end up in different clusters. The authors experimentally evaluate their approach against the *BalancedSummary* approach of [39] (on the TPC-E database) and show it has a higher accuracy. The work is continued in [82] where an approach is given to construct a summary given a set of required tables and a summary size. The authors show that this conditional clustering problem is NP-complete as well. A new entropy based distance function is defined, based on both intra- and inter-table column level relations; the distance between two tables is the total distance between the primary keys of the tables. The new distance function is experimentally shown to be better for conditional summaries than the distance described in [81].

An alternative, multi-clustering approach is described in [83]. The authors define three types of clustering approaches that could be used for summarization. The idea is that tables are first clustered using these types of clusterers and then a meta-clustering is applied to get the final results. The authors show that meta-clustering is also NP-complete. Finally some statistical techniques are presented to automatically increase the weighting of base clusterers that are more likely to be correct.

Both [81] and [83] also describe how a table could be chosen that is representative for all tables in a cluster. While in [83] the most central table is chosen using graph-based techniques, the weighted k-center approach described in [81] includes both centrality and (value based) importance of a table.

Approach

An approach that takes both entropy based importance and structural information into account while clustering tables into artifact schemas appears to be the best choice. The assumption here is that the most important tables likely contain the instance information of the artifacts. For this purpose entropy seems to be a good measure of importance, since it is higher when each record in a table contains more unique values. Since the authors of [81] describe such an approach this was used to calculate the base clusters. The approach described by the authors of [83] might provide better table clusters, but is computationally more complex. In addition the calculation of the main table in [83] does not take entropy based importance into account. Due to time constraints the alternative distance measures provided by [82] were not evaluated.

An naive approach for artifact schema identification would have been to first determine the most important tables and then assign all other tables to one or more of the most important tables. This schema identification approach ignores the fact that important tables that are close together are likely part of the same artifact though: the *request* table in the CD shops database might be considered at least as important as any table of the *CD* artifact schema, but it is still part of the *quote* artifact schema. The naive approach would generate a schema for the *CD* artifact if and only if a separate *quote* and *request* schema are generated, which is undesirable. Thus structural information needs to be taken into account as well.

None of the known schema summarization approaches assign tables to multiple clusters. Thus either an alternative clustering approach is required or the clusters generated by an existing summarization approach (the ***base clusters***) need to be expanded afterwards. For expansion a property needs to be available that can be used to determine if a table should be added to a base cluster. Artifact schema's have an interesting property that can be exploited for this purpose: artifact instance information is contained in the main table and other tables need to be included only if they contain information about an artifact instance. Thus to determine if a table needs to be included in an artifact schema only the relation between the table and the main table of an artifact schema needs to be investigated. This property will be used to expand base clusters using (indirect) foreign key references between the table and a main table.

For expansion of base clusters a level-wise algorithm was created that adds tables to each base cluster. It starts with an artifact schema that contains only the tables in the base cluster. At each level it evaluates foreign keys between tables in the artifact schema and dependent tables not in the schema: if these foreign keys are (indirectly) based on the primary key of the main table then the dependent tables are added to the schema. It stops if the maximum level to add is reached or if no tables were added at a given level.

Figure 13 shows an example of how this works based on the CD shop's example database described in subsection 1.5.3. In this example the base cluster consists of the *request*, *customer* and *cd_request* tables, with *request* being the main table. In the first iteration the foreign key between *request* and *quote* is evaluated: *reqid* is the referenced column and *quote_reqid* is the dependent column. Since *reqid* is the primary key of the main table the *quote* table is included. During the second iteration the *inclusion*, *cd_quote_order*, *reorder*, *quote_order*, *delivery* and *customer_payment* tables are added. The referenced column in all these cases is the *quote_reqid* column, which was shown to depend on the primary key of the *request* table during the previous iteration. During the third iteration the foreign key between *delivery* and *delivery_order* is evaluated. *Delivery_order* is not added, since the dependent column depends only the *delid* column which in turn does not depend on the primary key of the main table. Since no tables were added the algorithm stops.

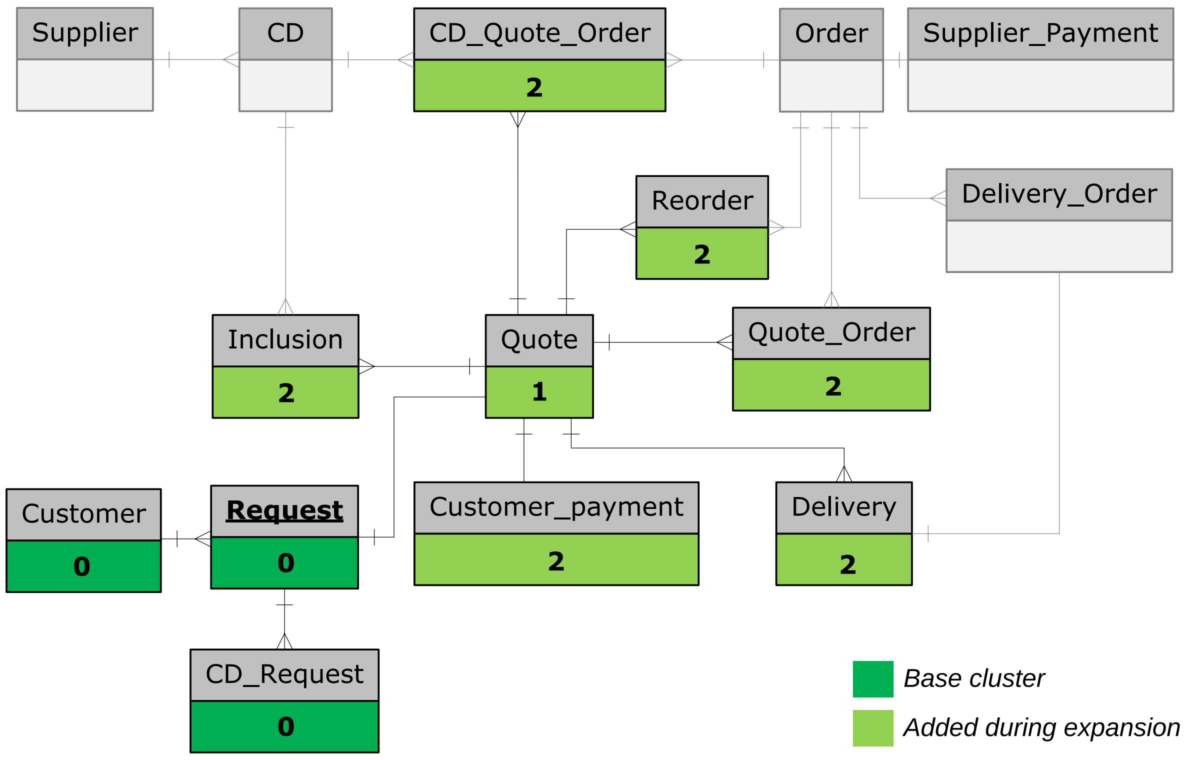


Figure 13: Schema expansion example

3.4 Conclusion

It is often not intuitively clear how the database of an ERP system is structured. Using the techniques in this chapter the database schema describing a structured dataset can be recovered. This includes the (re-)discovery of domain, primary key and foreign key meta data.

After a complete database schema is available the artifacts that are represented in the schema can be identified by applying a fuzzy clustering technique aimed at identifying artifact schemas. Thus we now have a number of artifact schemas clearly showing what artifacts are represented by the dataset. Since the lifecycle of each artifact is not yet known this is only a partial description of each artifact though.

Chapter 4 – Artifact lifecycle identification

The artifact schema does not explain an artifacts lifecycle, but the data described by it may be used to identify this lifecycle. Figure 14 recalls the steps that are required to discover an artifacts lifecycle given its schema and the corresponding data. This involves various control-flow discovery techniques, but these techniques require that lifecycle information is available in the form of an event log. Therefore the lifecycle information must first be extracted from all of the artifacts data.

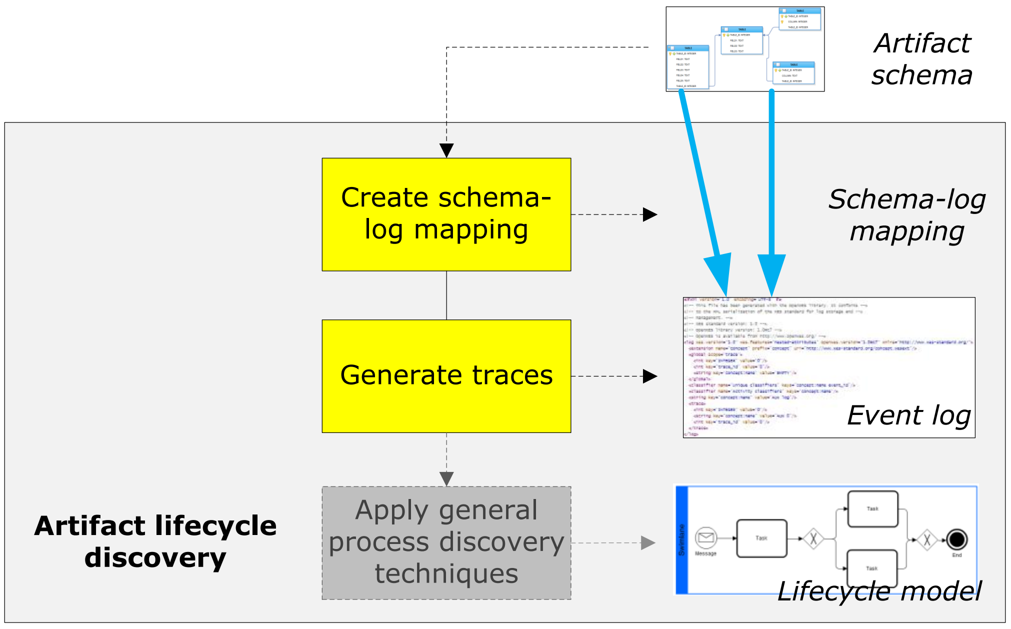

Artifact schema
Create schema-log mapping
Schema-log mapping
Generate traces
Event log
Artifact lifecycle discovery
Apply general process discovery techniques
Lifecycle model

Figure 14: Artifact lifecycle identification approach

Section 4.1 shows an overview of previous approaches to extract an event log from a source system. Section 4.2 then explains how a mapping from the artifact schema to an event log can be created automatically. The next section (4.3) shows how an event log can be extracted from the data using the created mapping. Finally in section 4.4 an overview of existing approaches to control flow discovery from event logs is presented, followed by a suggestion on how to use these to discover the lifecyle of an artifact.

4.1 Related work: Event log extraction

Event log extraction is the process of <u>extracting data from a (number of) source systems and converting this data into a format that is suitable for process mining</u>. The extraction of event logs can be separated into three steps: The creation of the mapping between the source systems data and an event log, the extraction of the raw data from the source system and finally the conversion of the raw data to an event log. Note that the first two steps may occur in reverse order (i.e. an analyst may receive a copy of the data as a first step) and that the final two steps may in practice be combined into a single step.

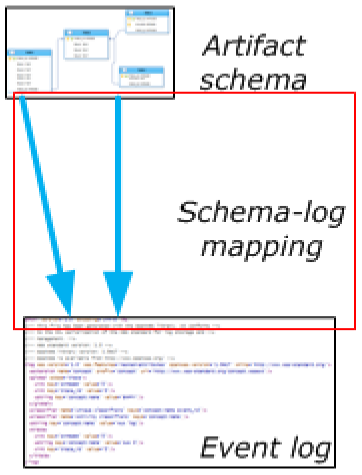
All known previous work on support for this step assumes that the mapping is created manually by domain experts [15–17], [21–23], [41–45], [84–87]. The majority of these papers describes how event logs can be extracted from a specific system but a few more general approaches have also been published [41], [42], [87], [88]. The most recent general approach is XESame, which is also the only general approach that supports the generation of event logs in the standard XES format (as described in section 2.3).

In general a number of challenges were given by previous work on event log extraction:

1. Due to convergence and divergence it is not always clear what the instance identifier should be or what events should be included in the log [15], [16], [41]. Traces should contain only events that belong to a single process [41], but convergence and divergence makes it harder to decide what that single process is.
2. What to include in the log depends on the goal of the process mining project [9], [41], [89]. This implies that there may not be a "one size fits all" approach for mapping the raw data to an event log.
3. Events should be at a consistent level of detail since they are treated equally during process mining [41], [89]. Similarly [16] mentions that in SAP multiple related records are stored with the same timestamp, implying that they might actually be related to the same event. A mapping method should take this into account.
4. The meaning of the (meta)data may not be clear when looking at it directly [15], [89], [90]. Examples of this are cryptic 4-letter names of SAP database tables or numeric values that encode a meaningfull label in the database. The general solution for this is to add ontological information to the log or simply replace cryptic values with a meaningfull label, but all of this requires domain knowledge.
5. Access to the (raw) data may be restricted, due to technical (e.g. proprietary data format, limited connectivity options), political or legal constraints [89].
6. Anonymization of for example names in the data may be required before the event log can be used [89].
7. Since event logs can be large, an extraction method should be able to scale sufficiently [41], [89].

4.2 Create schema-to-log mapping

The general problem of creating a mapping from a schema to an event log can be solved manually as done in previous work. One could simply first choose some columns to identify the instances, then some columns that specify event types and finally inspect every other column to see if it needs to be assigned as an attribute to an instance or event type. This information and the relations between all of them could then be entered into a tool such as XESame to complete the mapping creation. This is a time-consuming task which requires sufficient domain knowledge and an understanding of process mining. Therefore a level of automation is desirable.



During creation of the mapping challenge 1 through 4 as described in the previous section need to be taken into account. Aside from these some additional challenges can be identified:

8. Some events may be logged with limited time information (e.g. only date and no time information). In this case the order of events may not be sufficiently specific for process mining when only time information is used.
9. Some tables may contain many columns and/or be related to multiple event types. A mapping method should make sure that the right columns are mapped to the right event types (or attributes).

The artifact-centric approach already adresses the first two challenges of the previous section (as described in section 2.3). Challenge 3 can be addressed after the event log creation is completed by merging events in the log (using e.g. the approach described in [91]). Similarly challenge 4 can be addressed by inserting ontological information in the event log after its creation. Therefore the focus will be on the last two challenges.

4.2.1 Overal approach

Schema-to-log mappings can largely be identified automatically by a three-step approach based on timestamps and foreign key relations. Due to the earlier steps described in the previous chapter we can assume this information to be available. The first step of the approach is to identify event (type) columns based on their domain: Exactly one event should be created for each value in one of these columns. The remainining columns are then assigned as attributes to either the artifact instances or one or more event types. Finally the event type and attribute information is used to create an event mapping. This mapping can then be converted to a XESame mapping to allow for manual modification[5] or it can be used directly to generate event logs as described in section 4.3.

```
CreateTraceMapping ($\mathfrak{S}_A$)
Input: An artifact schema $\mathfrak{S}_A = (\mathfrak{T}_A, \mathfrak{F}_A, \mathfrak{D}_A, column\_domain, T_m)$
Terms: Set of event types $\mathbf{ET}$, event type ET, event table $T_{ET}$, event column $C_e$, set of event attribute
tables $\mathbf{T_{Event}}$, set of event attribute columns $\mathbf{C_a}$, set of instance attribute tables $\mathbf{T_{instance}}$ and a set of
instance attribute columns $\mathbf{C_A}$,

1.  $\mathbf{ET}$ = IdentifyEventTypes ($\mathfrak{S}_A$)

2.  $\mathbf{T_{instance}} = \{T_m\} \cup$ AllParents ($\{T_m\}, \mathfrak{S}_A$)
3.  $\mathbf{T_{instance}} = \mathbf{T_{instance}} \cup$ SelectInstanceChildTables ($T_m, \mathbf{T_{instance}}, \mathfrak{S}_A$)
4.  $\mathbf{C_A}$ = All non event columns $\in \mathbf{T_{instance}}$

5.  $\mathbf{EM} = \emptyset$
6.  For each ET = $(T_{ET}, C_e, \mathbf{C_a}) \in \mathbf{ET}$
7.      $\mathbf{T_{Event}} = \{T_{ET}\} \cup$ SelectEventChildTables($T_{ET}, \{T_{ET}\} \cup \mathbf{T_{instance}}, \mathfrak{S}_A$)
8.      $\mathbf{C_a}$ = All non event columns $\in \mathbf{T_{Event}} \setminus \mathbf{T_{instance}}$

9.      GeneralMappingItem ($\mathbf{C_{EID}}, T_{From}, \mathbf{F_{Link}}, \mathbf{AM_E}, \mathbf{LA_E}$) = CreateMapping ($T_m, T_{ET}, \mathbf{T_{Event}}, \mathbf{C_a}, \mathfrak{S}_A$)
10.     EM = (event column name, $\mathbf{C_{EID}}, C_e, T_{From}, \mathbf{F_{Link}}, \mathbf{AM_E}, \mathbf{LA_E}$)
11.     $\mathbf{EM} = \mathbf{EM} \cup \{EM\}$

12. GeneralMappingItem ($\mathbf{C_{TID}}, T_{From}, \mathbf{F_{Link}}, \mathbf{AM_T}, \mathbf{LA_T}$) = CreateMapping ($T_m, T_m, \mathbf{T_{instance}}, \mathbf{C_A}, \mathfrak{S}_A$)
13. TM = ($\mathbf{C_{TID}}, T_{From}, \mathbf{F_{Link}}, \mathbf{EM}, \mathbf{AM_T}, \mathbf{LA_T}$)

Output: A TraceMapping TM for the artifact
```

Figure 15: CreateTraceMapping algorithm

Figure 15 shows an overview of the *CreateTraceMapping* algorithm that is used to create a mapping from a schema to an event log. The details of the algorithm are explained in the subsections below. The identification of events (line 1) is further explained in subsection 4.2.2, the assignment of columns as attributes to traces (lines 2 to 4) and events (lines 7 and 8) is explained in subsection 4.2.3 and the actual creation of mappings for event types (lines 9 and 10) and the trace (lines 12 and 13) is explained in subsection 4.2.4.

The XES specific *LogMapping* (TM, **EX**, **CL, AG$_T$**, **AG$_E$**) is then constructed by combining various collected meta data and some default values:

- The TraceMapping TM is created as defined above.
- Extensions should be added as follows:
    - The default XES *concept* extension is required since it used for log, trace and event names.
    - The default XES *time* extension should be added if at least one event column has a timestamp domain.

[5] As described in Appendix C.

  - If some extensions were manually assigned to attributes then these would need to be added as well. This can be done by checking the created attribute mapping keys.

This creates the set of log extensions **EX**.

- Two classifiers are always available and should therefore be added:
  - The activity classifier (name: "Activity classifier", keys: "concept:name"), since all events have a unique type within the scope of a trace.
  - A unique classifier (name: "Unique classifier", keys: "concept:name eventID"), since the combination of the event type and eventID is unique in the entire log. Note that an eventID alone may not be unique since two event columns in the same table are assigned the same eventID.

  This creates the set of activity classifiers **CL**.
- Globals should be added based on the created trace and event mappings:
  - Each top-level attribute mapping or list attribute of a trace should be added as a global trace attribute. This creates the set of global trace attributes $\mathbf{AG_T}$.
  - The intersection of top-level attribute mappings and list attributes for all event should be added as global event attributes. This creates the set of global event attributes $\mathbf{AG_E}$.

At various places throughout the sections below the concepts of direct/all parent or child tables is used. The direct parent tables of table T are tables that are the parent table in a foreign key relation where T is the child table. All parent tables of T also include indirect parents, where there is an intermediate table that is the child of the parent, but also the parent of T. Child tables are similarly defined. Formally if **T** is a set of tables, T_i is a specific table, F is a foreign key as specified in section 2.1 and $\mathcal{F}_A$ the set of foreign keys in the artifact schema as specified in section 2.2 then formulas (2) to (5) define the direct and all parent and child tables. Figure 17 and 18 describe how the sets of all parent and child tables can be calculated.

$$DirectParents(\boldsymbol{T}) = \left\{T_p \mid \exists_{F=(T_p,\boldsymbol{C}_p,T_c,\boldsymbol{C}_c)\in\boldsymbol{F_A}}[T_c \in \boldsymbol{T}]\right\} \tag{2}$$
$$AllParents(\boldsymbol{T}) = DirectParents(\boldsymbol{T}) \cup AllParents(DirectParents(\boldsymbol{T})) \tag{3}$$

$$DirectChildren(\boldsymbol{T}) = \left\{T_c \mid \exists_{F=(T_p,\boldsymbol{C}_p,T_c,\boldsymbol{C}_c)\in\boldsymbol{F_A}}[T_p \in \boldsymbol{T}]\right\} \tag{4}$$
$$AllChildren(\boldsymbol{T}) = DirectChildren(\boldsymbol{T}) \cup AllChildren(DirectChildren(\boldsymbol{T})) \tag{5}$$

```
$AllParents\ (\mathbf{T}, \mathbb{S}_A)$
Input: A set of base tables $\mathbf{T}$ and artifact schema $\mathbb{S}_A$

1. $\mathbf{T}_{AllParents} = DirectParents\ (\mathbf{T}, \mathbb{S}_A)$
2. $\mathbf{T}_{AllParents} = \mathbf{T}_{AllParents} \cup AllParents\ (\mathbf{T}_{AllParents}, \mathbb{S}_A)$

Output: Set of all parent tables $\mathbf{T}_{AllParents}$
```

Figure 16: AllParents algorithm

```
$AllChildren\ (\mathbf{T}, \mathbb{S}_A)$
Input: A set of base tables $\mathbf{T}$ and artifact schema $\mathbb{S}_A$

1. $\mathbf{T}_{AllChildren} = DirectChildren\ (\mathbf{T}, \mathbb{S}_A)$
2. $\mathbf{T}_{AllChildren} = \mathbf{T}_{AllChildren} \cup AllChildren\ (\mathbf{T}_{AllChildren}, \mathbb{S}_A)$

Output: Set of all child tables $\mathbf{T}_{AllChildren}$
```

Figure 17: AllChildren algorithm

4.2.2 Identifying event types

All columns that are ordered by occurance and that are in tables for which each record is associated with exactly one artifact instance identifier are considered to represent a separate event type.

Aside from timestamps this includes colums of which the domain is marked as ordered by occurance. Domains can either be marked as such manually or a correlation analysis could be performed, comparing columns with a timestamp domain with columns with other domains in the same table. This makes it possible to include events for which limited time information is available.

If all correctly ordered columns were considered to be events (regardless of their relation with the artifact instance) then this would have included columns that might be identical for each instance (e.g. in parent tables of the main table). Although it is possible that these column values represent shared events this seems less likely, therefore these should be excluded. A simple approach for this is to exclude all columns in parent tables of the main artifact table T_m as specified by the *AllParents* function. A more elaborate approach is described in appendix D.II.

```
IdentifyEventTypes ($\mathfrak{S}_A$)
Input: A set of base tables $\mathbf{T}$ and artifact schema $\mathfrak{S}_A = (\mathfrak{T}_A, \mathfrak{F}_A, \mathfrak{D}_A, column\_domain, T_m)$

1. $\mathbf{T}_{PossibleEventTables} = \mathfrak{T}_A \setminus AllParents(\{T_m\}, \mathfrak{S}_A)$
2. $\mathbf{ET} = \emptyset$
3. For each table $T \in \mathbf{T}_{PossibleEventTables}$
4.   For each column $C \in T$
5.     If $column\_domain(C)$ is ordered by occurance Then
6.       Construct event type $ET = (T, C, \mathbf{C_a} = \emptyset)$ with event table T, event column C
         and a later to be identified set of attribute columns $\mathbf{C_a}$
7.       $\mathbf{ET} = \mathbf{ET} \cup \{ET\}$

Output: Set of all event types $\mathbf{ET}$
```

Figure 18: IdentifyEventTypes algorithm

The correctly ordered columns that are not explicitly excluded are called ***event columns***. For each of them an event type ET is constructed with event table T_{ET} and event column C_e. Event table T_{ET} is the table that contains C_e. The columns that contain attribute values will be associated with each event type as described in the next section. Figure 18 shows the complete algorithm to identify and construct all event types.

4.2.3 Assign attributes

All columns that are not considered to be events are assigned as attributes. These columns are assigned to the most specific event possible or as instance attributes if it is not possible to assign them to a specific event. For example: If an attribute column is part of a table without event columns, then it will be assigned to event columns in the parent table (assuming they exist). If there are event columns in the same table they will be assigned to those columns however. The assignment is done based on the table that contains the column:

- Columns in the set of artifact instance tables $\mathbf{T}_{instance}$ are assigned as instance attributes. The set of artifact instance tables consists of the main artifact table, all of its parents and all children that do not have another parent table with event columns.
- For each event type the columns in the corresponding set of event tables $\mathbf{T}_{Event}$ are assigned as event attributes. For event columns in the main artifact table there are no separate event attributes, thus then the set is empty. Otherwise the set consists of (1) the event table T_{ET}, (2) all child tables for which there is a

foreign key path from the event table to the child table that does not contain another event table and (3) all parent tables of the child tables that are not part of the set of instance tables $\mathbf{T}_{\mathbf{instance}}$ and do not have another event table as one of their parents. Note that the 2nd subset may contain tables that are also assigned to other event types.

```
SelectParentsWithoutEvents ($T_0$, $\mathbf{T}_{\mathrm{Ignore}}$, $\mathbb{S}_A$)
Input: Base table $T_0$, a set of tables to ignore $\mathbf{T}_{\mathrm{Ignore}}$ and artifact schema $\mathbb{S}_A$

1.  $\mathbf{T}_{\mathrm{valid}} = \emptyset$
2.  For each $T \in$ DirectParents ($T_0$, $\mathbb{S}_A$)
3.    If ($T \in \mathbf{T}_{\mathrm{Ignore}}$) Then
4.      Next T

5.    If $\neg$(Event columns in T) Then
6.      $\mathbf{T}_{\mathrm{Parents}}$ = SelectParentsWithoutEvents (T, $\mathbf{T}_{\mathrm{Ignore}}$, $\mathbb{S}_A$)
7.      If $\neg$(Parents with events found for T) Then
8.        $\mathbf{T}_{\mathrm{valid}} = \mathbf{T}_{\mathrm{valid}} \cup \{T\} \cup \mathbf{T}_{\mathrm{Parents}}$

Output: Set of valid parent tables $\mathbf{T}_{\mathrm{valid}}$
```

Figure 19: SelectParentsWithoutEvents algorithm

A key point are tables that are not child tables of the main artifact table or an event table. Since it is assumed that these tables also contain relevant information the columns in these tables should also be added as attributes to either events or the artifact instance. Figure 19 shows the *SelectParentsWithoutEvents* algorithm that can be used to find these types of tables given e.g. the main artifact table as input. Line 7 verifies that no parent tables that contain event columns were encountered in line 6. The algorithm is used when selecting tables that contain attributes for instances and event types as shown below.

```
SelectInstanceChildTables ($T_0$, $\mathbf{T}_{\mathrm{Ignore}}$, $\mathbb{S}_A$)
Input: Base table $T_0$, a set of tables to ignore $\mathbf{T}_{\mathrm{Ignore}}$ and artifact schema $\mathbb{S}_A$

1.  $\mathbf{T}_{\mathrm{valid}} = \emptyset$
2.  For each $T \in$ DirectChildren ($T_0$, $\mathbb{S}_A$)
3.    If $\neg$(Event columns in T) Then
4.      $\mathbf{T}_{\mathrm{Parents}}$ = SelectParentsWithoutEvents (T, $\mathbf{T}_{\mathbf{Ignore}}$, $\mathbb{S}_A$)
5.      If $\neg$(Parents with event columns found for T) Then
6.        $\mathbf{T}_{\mathrm{valid}} = \mathbf{T}_{\mathrm{valid}} \cup \{T\} \cup \mathbf{T}_{\mathrm{Parents}}$
7.        $\mathbf{T}_{\mathrm{valid}} = \mathbf{T}_{\mathrm{valid}} \cup$ SelectInstanceChildTables (T, $\mathbf{T}_{\mathrm{Ignore}}$, $\mathbb{S}_A$)

Output: Set of instance child tables $\mathbf{T}_{\mathrm{valid}}$
```

Figure 20: SelectInstanceChildTables algorithm

Figure 20 shows the *SelectInstanceChildTables* algorithm and how it can be used to determine the set of artifact instance tables $\mathbf{T}_{\mathbf{instance}}$. It starts by selecting all parent tables of the main artifact table T_m. It then selects all child tables of the main artifact table T_m until a table is encountered that contains event columns or have a parent table that contains event columns (unless that parent table is part of the artifact instance tables). This includes the selection of the parent tables of these child tables, since they cannot be assigned to any event type.

```
SelectEventChildTables ($T_0$, $\mathbf{T}_{Ignore}$, $\mathbb{S}_A$)
Input: Base table $T_0$, a set of tables to ignore $\mathbf{T}_{Ignore}$ and artifact schema $\mathbb{S}_A$

1.  $\mathbf{T}_{valid} = \emptyset$
2.  For each $T \in DirectChildren(T_0, \mathbb{S}_A)$
3.    If $\neg$(Event columns in T) Then
4.      $\mathbf{T}_{Parents} = SelectParentsWithoutEvents(T, \mathbf{T}_{Ignore}, \mathbb{S}_A)$
5.      If $\neg$(Parents with event columns found for T) Then
6.        $\mathbf{T}_{valid} = \mathbf{T}_{valid} \cup \mathbf{T}_{Parents}$
7.      $\mathbf{T}_{valid} = \mathbf{T}_{valid} \cup \{T\}$
8.      $\mathbf{T}_{valid} = \mathbf{T}_{valid} \cup SelectEventChildTables(T, \mathbf{T}_{Ignore}, \mathbb{S}_A)$

Output: Set of event child tables $\mathbf{T}_{valid}$
```

Figure 21: SelectEventChildTables algorithm

Figure 21 shows the *SelectEventChildTables* algorithm and how it can be used determine the set of attribute tables $\mathbf{T}_{Event}$ for each event type. It is similar to the algorithm for instance attributes, except that it selects all child tables until a table that contains event columns is encountered. Thus child tables that have other event types as parents are still selected. In addition to this the parent tables of the selected child tables are selected, unless the selected child table has a parent table that contains events. Note that parent tables can contain events if they are part of the instance attribute tables, since that implies that these parent tables are not assigned to another event type. Parent tables that are already part of the instance attribute tables are never selected.

The running time of both the *SelectInstanceChildTables* and *SelectEventChildTables* algorithm are polynomial with respect to the number of tables and columns in each table. If $|\mathbf{C}|_{max}$ is used to denote the maximum number of columns in a table then the worst case running time is $O(|\mathcal{T}_A| \cdot (|\mathbf{C}|_{max} + |\mathcal{T}_A| \cdot |\mathbf{C}|_{max}))$, since for each table it needs to be verified if the columns are event columns and in the worst case scenario the same needs to be done for all other tables. This scenario assumes that all other tables are parent tables of the table. The worst running time of the algorithm can be improved to $O(|\mathcal{T}_A| \cdot (|\mathbf{C}|_{max} + |\mathcal{T}_A|))$ however by keeping track of the validity of tables as described in appendix D.I.

4.2.4 Creating the mapping

Given that the event types are known and the attribute columns are selected all information is available to create a schema-to-log mapping as described in section 2.3. This mapping consists of a *LogMapping* (including its associated globals, extensions and classifiers), a *TraceMapping* and a number of *EventMappings*. The creation of the TraceMapping and the EventMappings is similar; for the attribute instance a TraceMapping is created while for each event type an EventMapping is created. The LogMapping can then be created using information gathered during the creation of these other mappings and techniques taken from XESame [41].

```
*CreateMapping* ($T_m$, $T_0$, $\mathbf{T}_{Attr}$, $\mathbf{C}_{Attr}$, $\mathfrak{S}_A$)
***Input:*** Main artifact table $T_m$, base table $T_0$ (with primary key $\mathbf{C}_{p0}$), a set of
attribute tables $\mathbf{T}_{Attr}$, a set of attribute columns $\mathbf{C}_{Attr}$ and artifact schema $\mathfrak{S}_A$

1.  $\mathbf{C}_{ID} = \mathbf{C}_{p0}$
2.  $T_{From} = T_m$
3.  $\mathbf{F}_{Link} = Path(T_m, T_0)$
4.  $\mathbf{AM} = \emptyset$
5.  $\mathbf{LA} = \emptyset$
5.  $\mathbf{T}_{one2one}, \mathbf{T}_{one2many} = SplitOneAndMany(T_0, \mathfrak{S}_A)$
6.  For each $T \in (\mathbf{T}_{one2one} \cap \mathbf{T}_{Attr}) \cup \{T_0\}$
7.    $\mathbf{F}_{Link} = \mathbf{F}_{Link} \cup Path(T_0, T)$
8.    For each $C \in \mathbf{C} \cap \mathbf{C}_{Attr}$
9.      $AM$ = Create attribute mapping for $C$
10.     $\mathbf{AM} = \mathbf{AM} \cup \{AM\}$

11. $\mathbf{T}_{Attr} = \mathbf{T}_{Attr} \setminus \mathbf{T}_{one2one}$
12. For each $T \in \mathbf{T}_{one2many}$
13.   If $T \in \mathbf{T}_{Attr} \vee (\mathbf{T}_{Attr} \cap AllChildren(T, \mathfrak{S}_A) \neq \emptyset)$ Then
14.     $LA = CreateMapping(T_m, T, \mathbf{T}_{Attr}, \mathbf{C}_{Attr}, \mathfrak{S}_A)$
15.     $\mathbf{LA} = \mathbf{LA} \cup LA$

***Output:*** GeneralMappingItem ($\mathbf{C}_{ID}$, $T_{From}$, $\mathbf{F}_{Link}$, $\mathbf{AM}$, $\mathbf{LA}$)
```

Figure 22: CreateMapping algorithm

Figure 22 shows the *CreateMapping* algorithm that can be used to create a GeneralMappingItem ($\mathbf{C}_{ID}$, T_{From}, $\mathbf{F}_{Link}$, **AM**, **LA**) that serves as the basis for a TraceMapping, EventMapping or ListAttribute. The basic idea is that each created mapping consists of a set of tables for which only one record exists for each record in the chosen ***base table*** T_0, thus ensuring that multiple values do not occur. The algorithm starts by splitting the given attribute tables $\mathbf{T}_{Attr}$ into a set for which this condition holds $\mathbf{T}_{one2one}$ and a set of attribute tables for which this condition does not hold $\mathbf{T}_{one2many}$. This is done by the *SplitOneAndMany* algorithm described in Figure 23. One mapping is then created for the base table and all tables in $\mathbf{T}_{one2one}$. This mapping contains a number of submappings (or *ListAttributes*) **LA** as required for the tables in $\mathbf{T}_{one2many}$. Note that to create an EventMapping the event column C_e and event name[6] needs to be added to the resulting GeneralMappingItem (as shown on line 10 of Figure 15), while to create a TraceMapping the set of event mappings EM needs to be added (as shown on line 13 of Figure 15).

The algorithm is initalized with the main artifact table T_m, a base table T_0 with primary key $\mathbf{C}_{p0}$, a set of attribute tables $\mathbf{T}_{Attr}$, a set of attribute columns $\mathbf{C}_{Attr}$ and the artifact schema information $\mathfrak{S}_A$. The set of attribute columns $\mathbf{C}_{Attr}$ consists of all columns in the attribute tables, except for the event columns and duplicates that may have been pruned based on foreign keys. For a TraceMapping the base table is the main artifact table T_m and the attribute tables are the artifact instance tables $\mathbf{T}_{instance}$. For an EventMapping the base table is the event table T_{ET} and the attribute tables are the event attribute tables $\mathbf{T}_{Event}$. The *Path* between two tables consists of the sequence of foreign keys connecting those tables; it can be calculated using e.g. Dijkstra's algorithm [92].

[6] The event type names can be trivially generated using a combination of the event table and event column names. Heuristics can be used to remove time or duplicate terms.

```
SplitOneAndMany ($T_0$, $\mathfrak{S}_A$)
Input: Base table $T_0$ and artifact schema $\mathfrak{S}_A$
1.  $\mathbf{T}_{one2one} = \emptyset$
2.  $\mathbf{T}_{one2many} = \emptyset$
3.  For each $F = (T_p, \mathbf{C}_p, T_c, \mathbf{C}_c) \in$ ChildForeignKeys ($T_0$, $\mathfrak{S}_A$)
4.    If F is one-to-one Then
5.      $\mathbf{T}_{one2oneChild}, \mathbf{T}_{one2manyChild} =$ SplitOneAndMany ($T_c$, $\mathfrak{S}_A$)
6.      $\mathbf{T}_{one2one} = \mathbf{T}_{one2one} \cup \{T_c\} \cup \mathbf{T}_{one2oneChild}$
7.      $\mathbf{T}_{one2many} = \mathbf{T}_{one2many} \cup \mathbf{T}_{one2manyChild}$
8.    Else
9.      $\mathbf{T}_{one2many} = \mathbf{T}_{one2many} \cup \{T_c\}$

10. For each $F = (T_p, \mathbf{C}_p, T_c, \mathbf{C}_c) \in$ ParentForeignKeys ($T_0$, $\mathfrak{S}_A$)
11.   If $T_p$ was not evaluated before Then
12.     $\mathbf{T}_{one2oneParent}, \mathbf{T}_{one2manyParent} =$ SplitOneAndMany ($T_p$, $\mathfrak{S}_A$)
13.     $\mathbf{T}_{one2one} = \mathbf{T}_{one2one} \cup \{T_p\} \cup \mathbf{T}_{one2oneParent}$
14.     $\mathbf{T}_{one2many} = \mathbf{T}_{one2many} \cup \mathbf{T}_{one2manyParent}$

Output: A set of tables with one record per record in the base table $\mathbf{T}_{one2one}$ and
a set of tables with multiple records per record in the base table $\mathbf{T}_{one2many}$
```

Figure 23: SplitOneAndMany algorithm

The *SplitOneAndMany* algorithm classifies all tables connected to the given base table T_0 as having one record per record in the base table (the $\mathbf{T}_{one2one}$ set) or as having multiple records per record in the base table (the $\mathbf{T}_{one2many}$ set). As shown in Figure 23 this is done by recursively verififying foreign keys in the child direction (where T_0 is the parent table) and the parent direction (where T_0 is the child table). In the parent direction there will always be only one record for each record in the base table. In the child direction it has to be checked if more records exist in the child table for each record in the base table. This can be done by calculating the matched average fanout between the two tables as defined in the related work subsection of section 3.3.

Although not shown in the *CreateMapping* algorithm some care needs to be taken when automatically constructing *attribute mappings* since each attribute needs to have a unique key in XES[7]. Without loss of generality it is assumed that a default key is assigned to each domain and that this key should be used if possible. This may result in multiple attribute mappings with the same key. To resolve this one top-level attribute mapping should be constructed with the default domain key with some default value (e.g. "multiple"). Each attribute mapping with that key should then be placed below the top-level attribute with a new unique key (generated using e.g. the table and column names).

4.2.5 Mapping the CD shop quote artifact

To further explain the algorithm this subsection will show how the mapping is created for the *quote* artifact of the CD shop example introduced in subsection 0. The resulting log mapping is available in Appendix E in XML format. For simplicity we will use F=(*parent table*, *child table*) to denote a foreign key between two tables in this subsection.

To identify event types first the parent tables of the main *quote* table have to be identified: These are the *request* and *customer* table according to the definition given by (3). Note that this does not include the *cd_request* table[8]. The remaining possible event

[7] The same point of attention (and solution) also holds for list attributes. For list attributes the default key value can be the name of the base table.
[8] This table could also be excluded as an event table by removing all children of parent tables that are not children of the main table. The basic idea is the same however.

tables $\mathbf{T}_{\mathbf{PossibleEventTables}}$ are then *quote*, *cd_request*, cd_*quote_order*, *inclusion*, *reorder*, *quote_order*, *customer_payment* and *delivery*. For this example only columns with a timestamp datatype are considered to be event types; therefore an event type is constructed for the *date_invoice_issue*, *date_payment_sent*, *date_payment_received* (all from *customer_payment*), *customer_accept_shipment_date* (from *delivery*) , *adding_date* (from *quote_order*), *opening_date*, *acceptance_quote_date*, *rejection_quote_date*, *customer_no_deliverable_notification_date* (all from *quote*) and *reorder_date* (from *reorder*) columns.

The set of tables that includes attributes for artifact instances $\mathbf{T}_{\mathbf{instance}}$ is then determined in two steps. First the *quote* main table and its parents *request* and *customer* are added. Then the children of these tables are inspected for event columns; since *cd_request*, *cd_quote_order* and *inclusion* do not contain event columns these are added as well. Thus in total there are 6 tables in $\mathbf{T}_{\mathbf{instance}}$. All non-event columns in $\mathbf{T}_{\mathbf{instance}}$ are assigned as attribute columns to the artifact instance.

Except for the event columns in the *quote* table all event columns are in a table without child tables. The children of the *quote* table either contain event columns or they were already assigned to the artifact instance, thus no attributes are assigned to event types from this table (e.g. *opening_date*). For the other event types all non-event columns in the event table are assigned; for all 3 event types from the *customer_payment* table the *price* column is added for example.

Since all event types only have attribute columns from the event table (in which the event column is defined) the creation of an EventMapping is similar for all of them. For the *date_invoice_issue* event column from the *customer_payment* event table this means that $\mathbf{C}_{\mathbf{EID}}$ = {quote_reqid} (the primary key of *customer_payment*) , T_{From} = *customer_payment*, $\mathbf{F}_{\mathbf{Link}}$ = {(*quote*, *customer_payment*)}, $\mathbf{AM}_{\mathbf{E}}$ = {("price", integer, price)} and $\mathbf{LA}_{\mathbf{E}} = \emptyset$. Here (*quote*, *customer_payment*) is added as a link between the main artifact table and the event table.

The creation of the trace mapping is slightly more complex, since there are child attribute tables that have multiple values for each instance. Looking at the schema we note that $\mathbf{T}_{\mathbf{one2one}}$ = {*request*, *customer*, *customer_payment*} and $\mathbf{T}_{\mathbf{one2many}}$ = {*cd_quote_order*, *reorder*, *cd_request*, *inclusion*, *quote_order*, *delivery*}. Since *customer_payment* is not in the set of artifact instance tables, this means that only *request* and *customer* need to be added to the basic trace mapping. Thus $\mathbf{C}_{\mathbf{TID}}$ = {reqid} (the primary key of *quote*) , T_{From} = *quote*, $\mathbf{F}_{\mathbf{Link}}$ = {(*request*, *quote*), (*customer*, *request*)}, $\mathbf{AM}_{\mathbf{T}}$ = all non-event columns from *quote*, *customer* and *request*.
There are 3 artifact instance tables with multiple values (in $\mathbf{T}_{\mathbf{instance}} \cap \mathbf{T}_{\mathbf{one2many}}$): *cd_request*, *cd_quote_order* and *inclusion*. For these 3 a list attribute needs to be created to store the list of multiple values. Since neither of the 3 has further child tables this is done similar to the event mappings above. For the *inclusion* table this results in $\mathbf{C}_{\mathbf{ID}}$ = {*quote_reqid*, *cd_name*} (the primary key of *inclusion*) , T_{From} = *inclusion*, $\mathbf{F}_{\mathbf{Link}}$ = {(*quote*, *inclusion*)}, **AM** = {("quantity", integer, *quantity*)}.

4.3 Generating traces

Since we now have a mapping an event log can be generated from the dataset. This involves the extraction of the data from the source system and the conversion to the XES format according to the defined mapping. A generic solution for this process is XESame, but due to some limitations this cannot be used to extract the log according to the defined mapping. Therefore the approach used in XESame is adapted to handle the mapping defined in section 2.3.

Schema-log mapping
Artifact schema
Event log

4.3.1 Evaluation of previous approaches

The behaviour of artifacts is described as much by their inner lifecycle as by the interaction with other artifacts [93]. To be able to analyze this using process mining techniques this interaction must be available in event logs. This can be done by adding eventId's and instanceId's of other artifacts as attributes to instances and events. Due to divergence these attributes often have multiple values, e.g. multiple *quote* id's should be listed for the creation event of a single *order*. Unfortunately XESame does not support the creation of traces or events with multi-valued attributes [41] and conversations with the author showed that significant architectural changes would be required to add this functionality. Therefore XESame could not be used directly for log extraction. Since no other generic extraction tools exist to create an event log in the XES format an alternative approach was required.

4.3.2 Approach

The three-step approach described in [41] was used as a basis for the approach. These three steps were (1) construction of queries to extract data, (2) executing the queries to populate a cache database and (3) create a XES event log from the cache database [41]. In addition to this large parts of the extraction process of XESame were reused. The most important changes were in step 1 and 2. In step 1 query generation was modified to support the modified mapping definition, including the addition of query generation for any level of depth of ListAttributes. In step 2 processing of queries was also modified to support any level of depth of ListAttributes. This included making modifications to the cache database as shown below. In step 3 some minor modifications were made to adapt for the modified cache database.

In the original XESame approach queries were only generated and executed at the trace and event level. First traces are exported to the cache database and then events are extracted and added to these traces. The matching of traces with events is done using the user-specified *trace_id*, which can be a combination of columns. To extract lists of attribute values the query construction and execution for ListAttributes needs to be done in a similar way as that for events. An *event_id* and *attribute_id* were thus added to the cache to allow for unique identification of events and attributes respectively. The values for these were constructed by combining the name or key with the identifying columns, resulting in values that are unique in the log. Suppose for example that the "quote order adding" event is identified by the *quote__req_id* and *order_order_id* columns then the generated event_id would be "quote_order_adding_[quote_req_id]_[order_order_id]".

Figure 24 shows the modified cache database. The following changes were made:

- A *log_id* column was added to all tables to make log specific identifiers (such as *event_id*) unique. This makes it possible extract multiple logs to the cache database before starting the extraction process, thus making it easier to handle multiple artifact logs at the same time.
- A text-based *event_id* column was added to be able to identify parent events when extracting ListAttributes. The original *event_id* column was renamed to *id*.
- A text-based *attribute_id* column was added to be able to identify parent attributes when extracting ListAttributes.
- A generated integer *id* column was added to the XTrace and XEvent tables to make it easier to reference these tables from XAttribute table (considering that e.g. the event_id column by itself was not sufficient to uniquely identify events anymore).
- Extensions, globals and classifiers were added to the cache database. By design these log properties were calculated using the provided XESame mapping when an event log was extracted. Since this made it harder to reuse the part of XESame starting from the cache database this was changed to allow for a clearer separation of step 2 (source database to cache) and step 3 (cache to XES file).

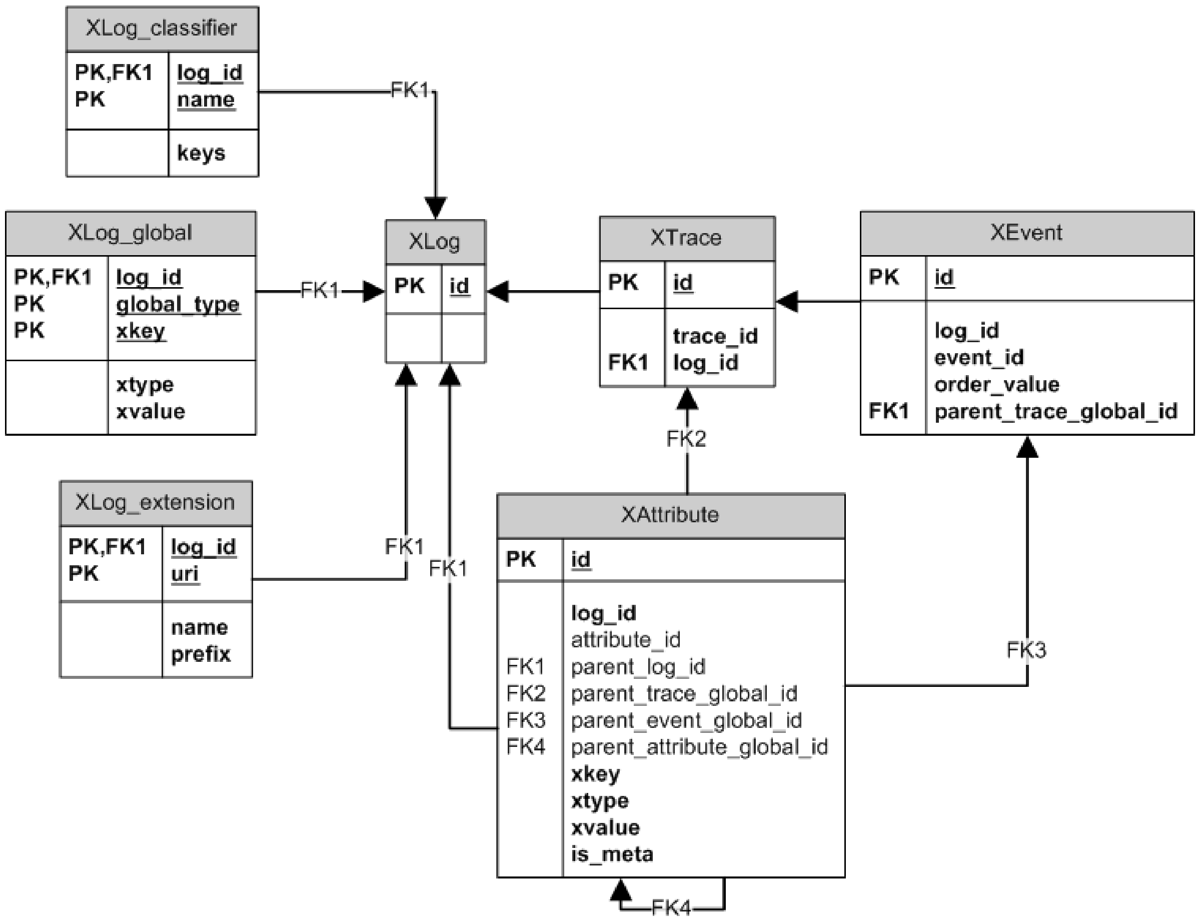
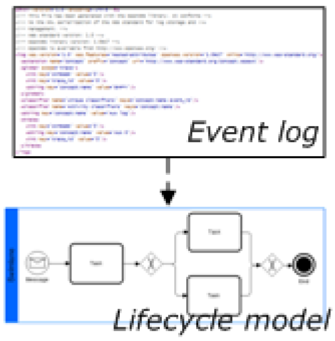


Figure 24: Cache database

4.4 Event log based artifact lifecycle discovery



The goal of artifact lifecycle discovery is to discover both the internal lifecycle of an artifact and its interaction with other artifacts, thereby fully describing how an artifact operates. There is no previous work on (semi-)automated interaction discovery and it is beyond the scope of this project to develop such a method. Therefore this section is limited to a brief overview of existing approaches to discover the control flow of a process and how to apply these to discover the inner lifecycle of an artifact.

4.4.1 Related work: Control flow discovery

Control flow discovery is the construction of a control flow model using an event log as input. Since the pioneering work in the context of software engineering described in [94] over 45 algorithms have been created for this purpose [95]. The majority of these focus on accurately describing the behaviour captured in the log, but this may result in unreadable spaghetti models if the execution of the business process is not explicitly controlled [89]. Due to this recently some algorithms have been proposed that balance between readability and accurately reflecting the behaviour in the log [91], [96]. These fuzzy algorithms simplify the discovered control flow model by merging fine-grained activities together to form coarse-grained activities.

The genetic mining algorithm was created to accurately describe any kind of behaviour found in an event log [97], [98]. It was shown to tackle most control flow discovery challenges in [95]. Its main disadvantage is its computation time, but a recent distributed version of the algorithm makes it possible to finish in 10% to 25% of the original time by using more CPU resources [99].

A number of preprocessing steps have been proposed to handle complex event logs. Trace clustering [100] splits an event log into homogenous groups of traces, thus resulting in groups of traces that can each be represented by a relatively simple model. Trace alignment [101] preprocesses the traces, such that exceptional behaviour is highlighted.

All of the algorithms described in this section are implemented in ProM. ProM is an actively maintained plugable open-source framework to execute process mining analysis [88]. It as available at www.promtools.org/prom6/. ProM is also used extensively in the process diagnostics method described in [102]. This method describes 5 steps one should follow when analyzing a process with process mining techniques: (1) log preparation, (2) log inspection, (3) control flow analysis, (4) performance analysis and (5) role analysis.

4.4.2 Approach

Assuming it is unknown if the lifecycle of the artifact is explicitly controlled a structured trial and error approach is recommended. First one should inspect the log and remove incomplete traces as described in [102]. Then the detailed lifecycle of the process should be discovered using the genetic algorithm. If the artifact turns out to have a nicely structured internal lifecycle then this step is finished. Since this may not be the case the fuzzy miner could be used to get a more readable version of the internal lifecycle of the artifact. In this case the trace clustering and alignment approaches could also be used to simplify the event log, thereby increasing the chances of getting a readable version of the artifacts internal lifecycle using the genetic or fuzzy miner.

Figure 25 and Figure 26 show the results of applying the genetic miner[9] to the event logs generated for the *quote* and *order* artifacts. They were created with the default settings of the plug-in, without any preprocessing of the event logs. Note that the discovered lifecycles are quite close to the lifecycles shown in Figure 8, implying that the method as a whole worked quite well.

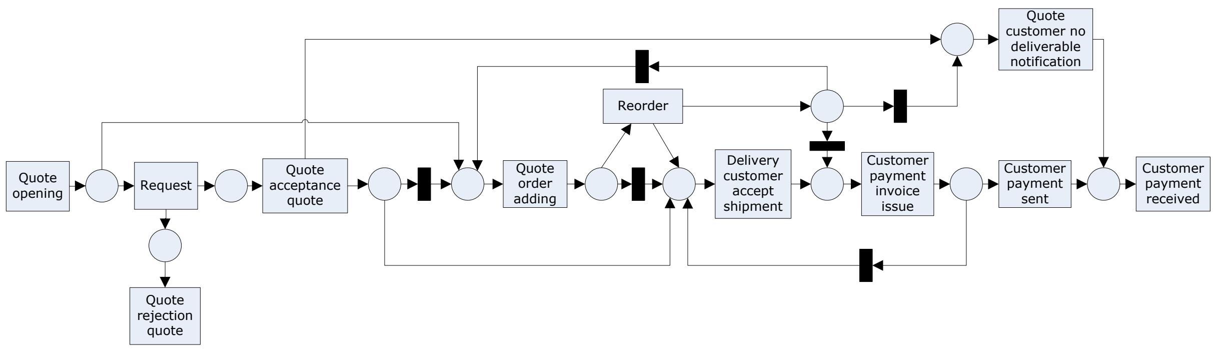

Quote opening
Request
Quote rejection quote
Quote acceptance quote
Quote order adding
Reorder
Delivery customer accept shipment
Customer payment invoice issue
Quote customer no deliverable notification
Customer payment sent
Customer payment received

Figure 25: Discovered Quote lifecycle

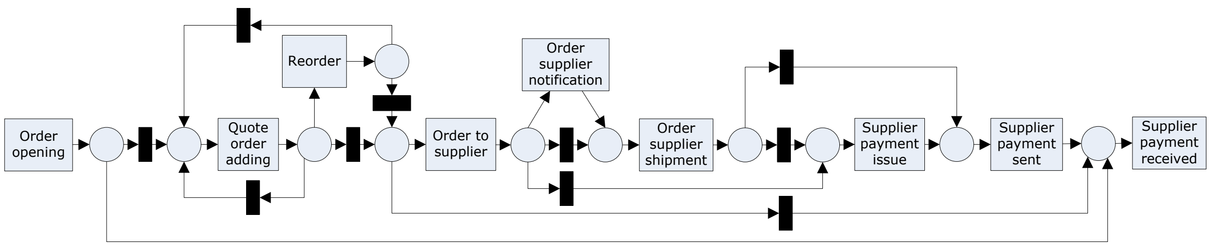


Figure 26: Discovered Order lifecycle

[9] "Mine Heuristic Net using Genetic Miner" plug-in created by A.K. Alves de Medeiros

4.5 Conclusion

The artifact schema and the data described by it can be used to (re-)discover the lifecycle of an artifact. As described in this chapter first a mapping from the schema to an event log needs to be identified. The creation of this mapping can be done automatically by first identifying the available event types, then assigning columns as attributes to each event type and the artifact instance and finally combining this information to create a schema-to-log mapping.

The schema-to-log mapping can then be used to extract an event log from the dataset, followed by the discovery of the inner lifecycle of the artifact using existing control flow discovery techniques. When the discovered lifecycle is combined with the artifact schema identified in Chapter 3 the description of the artifact is complete.

Chapter 5 – Empirical evaluation

The approach described in the previous chapters was evaluated using a prototype implementation. The techniques were evaluated using an artificial dataset based on the CD shop example and two large real-life datasets provided by a large food wholesale and retail company. Section 5.1 describes the datasets in more detail. The remaining sections describe the experiments for each of the techniques, following the order in which they were presented before.

All experiments were executed on a laptop with an Intel core i3 2,13 Ghz processor with PostgreSQL 9.0 as the underlying database. The prototype was implemented in Java since a large number of relevant libraries and frameworks (e.g. OpenXES[10]) were already available in Java. An overview of the prototype is available in Appendix F.

5.1 Dataset descriptions

Three different datasets were used for the experiments. The main statistics of these datasets are shown in Table 3. Below the table a description of the datasets is given. Note that for both real-life datasets a number of tables were not in use; whenever these statistics are used in the remainder of this chapter only the tables that are in use are taken into account.

Name	*Number of tables*	*Average column count*	*Max. column count*	*Average row count*	*Maximum row count*	*Size*
CD shop	16	3.31	6	257	681	8.7 MB
Article maintenance	365	13.54	195	523 726	85 969 702	40.5 GB
Article maint. (*in use*)	317	14.16	195	603 029	85 969 702	40.5 GB
ERP system (*complete*)	3 390	18.09	2 415	293 368	116 489 821	217 GB
ERP system *(in use)*	1 451	14.98	420	685 401	116 489 821	217 GB

Table 3: Dataset statistics

The ***CD shop dataset*** is an artificial dataset originally created to validate the research described in [103]. It was generated by modeling the CD shop as an artifact-centric system in CPN tools[11] and running the model as a simulation. The database schema is identical to the schema described in subsection 1.5.3.

The ***article maintenance dataset*** is a real-life dataset taken from the product management system of a large food wholesale and retail company. The product management system was developed in-house and uses the DB2 relational database system to store its data. Aside from basic product information such as names and product prices it also contains associated data such as the selection of products available for specific stores. The dataset consists of a copy of the complete database, including a number of tables that were not in use.

The ***ERP system dataset*** is a real-life dataset provided by the same food wholesale and retail company. It is a copy of the main database that supports the ERP system that was developed in-house over the last 30 years. Due to the architecture of the system a large number of tables are present in the database that are not in use. In addition to this no primary key is defined for about half of the tables and no foreign keys are enforced in the database.

[10] A reference implementation of the XES standard.
[11] http://cpntools.org/

5.2 Artifact schema identification

5.2.1 Schema extraction

Domain extraction

Experimental evaluation of the domain extraction technique showed that the two-step approach was a good choice. The experiments were executed on both the CD shop and article maintenance dataset. Table 4 shows the run times of the experiments. Lines in the table denoted with (XES) are results for the heuristic first step, while the (DBscan…) results are for the second step. For the article maintenance dataset the results for the second step are shown for each of the XES datatypes seperately.

Description	*Tables*	*Columns*	*Run time*
CD shop (XES)	16	53	< 1 s
CD shop (DBscan with min-hash on all columns)	16	53	3.3 s
Article maintenance (XES)	317	4 488	26.5 s
Article maintenance (DBscan w. min-hash on Integers)	297	1201	25 m 41 s
Article maintenance (DBscan w. min-hash on Booleans)	241	713	8 m 38 s
Article maintenance (DBscan w. min-hash on Dates)	311	719	> 15 h

Table 4: Domain extraction experiment statistics

The experimental results show that clustering with DBscan indeed scales more than linear with the number of columns, while heuristics that can drastically reduce this number can be executed in a relatively short amount of time. Even with the use of these heuristics the set of columns with timestamp (Date) values still proved to be too big to cluster in a reasonable amount of time however.

The precision of the approach could be improved significantly however. The first step correctly classifies most data according to the XES datatype (some exceptions are explained below), but it proved to be impossible to select a set of parameters for the second step that resulted in a set of meaningfull clusters. In the approach it was assumed that the second step could be executed with a single set of parameters for the clustering algorithm. For the CD shop experiments were executed with both the min-hash and PCA methods. These experiments showed that a single set of parameters was not feasible, since parameters that resulted in a relatively meaningfull set of clusters for string values were incorrect for for example integer values (and vice versa). Therefore a likely improvement would be to use a different clustering approach for each of the data types in the second step.

An attempt was made to also apply the approach on the ERP system dataset, but this had to be abandoned. The most significant problem was that all timestamp and floating point values were stored as integer values. Thus heuristics to separate the different data types only resulted in a set of string and integer columns. Since these sets were too big to handle with traditional clustering approaches no further progress could be made.

Primary key extraction

To evaluate the primary key extraction approach a number of experiments were executed on tables for which a primary key was defined. The experiments were run with a sample of 1 000 records for the Gordian algorithm and a sample of 5 000 records for the HCA algorithm. For the CD shop and article maintenance datasets two runs were executed: a run in which only the most likely primary key was requested and a run in which the most likely two primary keys were requested. For the ERP system dataset only a run in which the most likely primary key was requested was execiuted. The results are shown in Table 5. Here ***precision*** is defined as the number of correct results divided by the number of results (found), while ***recall*** is defined as the number of correct results divided by the existing (defined) number of primary keys.

Description	*Run time*	*PK's defined*	*PK's found*	*PK's correct*	*Precision*	*Recall*
CD shop (1st key)	25s	16	16	13	0.81	0.81
CD shop (2 keys)	26s	16	25	15	0.60	0.94
Article maintenance (1st key)	4h 34m	317	309	240	0.78	0.76
Article maintenance (2 keys)	9h 52m	317	555	255	0.46	0.80
ERP system (1st key)	42h 52m	716	645	403	0.62	0.56

Table 5: Primary key experiment statistics

The exponentional nature of the problem is apparent when comparing the runtimes of the single key article maintenance and ERP system experiments. Although the ERP system dataset is less than 3 times the size of the article maintenance dataset with respect to this experiment the runtime of the ERP system dataset is over 9 times as high. The most likely explanation is that this is caused by the higher number of columns in the tables of the ERP systems dataset.

During the experiments is also became clear that the total runtime depends highly on a limited number of tables. For both the article maintenance and ERP system dataset over half of the run time was used for approximately 10 tables. Initially these tables used a significantly larger portion of the run time, but this was resolved by cancelling steps in the algorithm after a predetermined amount of time. Both the Gordian and HCA part were cancelled after 30 minutes, while the verification step was cancelled after 1 hour. Note that the algorithm could still continue if the Gordian part was cancelled, but not if the HCA or verification step were cancelled.

Although both precision and recall score quite well when only the most likely primary key is retrieved improvements are possible. Analysis of the incorrect results showed that this was usually a column containing timestamp values. Since timestamp values have a high chance of being unique these columns could be used as a single column key, whereas a composite primary key was defined in these cases. Therefore improvements could be made by taking the datatype of the column into account or by placing less importance on the number of columns in the key.

Finally it is noteworthy to mention that the algorithm was also used to detect candidate keys in the 735 tables of the ERP system dataset for which no key was defined. This allowed the detection of keys with up to 117 columns. For this purpose a number of records where allowed to violate the primary key in the verification step. For 495 tables a candidate key was detected if 10 records were allowed to violate the key, which was 20 more than when no violations were allowed. Unfortunately the HCA-Gordian algorithm did not allow for the discovery of so-called soft candidate keys (i.e. keys for which some fraction of the data is invalid), thus it is unknown if likely candidate keys were removed prior to the verification step.

Foreign key extraction

The experimental evaluation of foreign key extraction showed that this was indeed not trivial. Experiments were run with 1 and 2 column combinations for both the CD shop and article maintenance datasets. An attempt to run a 1 column experiment with the ERP system dataset failed due to a lack of memory of the Java virtual machine. The run times of the experiments are shown in Table 6. The *IND candidates* column shows the number of IND candidates after pruning took place. Aside from the experiments shown in the table an attempt was also made to extract 2 column foreign keys from the article maintenance dataset using the original SPIDER algorithm, but since progress was still very low after some days this was abandoned.

Description	*Total run time*	*Indexing time*	*IND candidates*	*IND's*
CD shop (1 column)	6 s	3 s	162	39
CD shop (2 columns)	2 s	1 s	52	3
Article maintenance (1 column)	4h 55m	2h 52m	139 225	16 335
Article maintenance (2 columns)	6d 8h 28m	22h 32m	189 340	N/A[12]

Table 6: Foreign key experiment statistics

The large increase in running time between the 1 column and 2 column article maintenance experiments can be explained by the increase in value combinations that need to be verified; this also increases exponentially with the number of columns. Thus although previous research shows that it should be possible to use the SPIDER algorithm with data sets of this size the scalability appears to be bounded.

A comparison between the Dice coefficient and the longest common subsequence (LCS) of the name showed that the LCS performed favorably. The main use of both turned out to be as a discriminator when a column combination is included in multiple parent column combinations. For the CD shop dataset the IND with the highest LCS name score was always the foreign key (if one existed), while the Dice coefficient falsely classified 2 foreign keys as incorrect. Both correctly classified 12 IND's as incorrect foreign keys, thereby eliminating a large number of false positives. For the article maintenance dataset the same strategy could be followed: The LCS falsely classified 10 foreign keys as incorrect, while the Dice coefficient falsely classified 25 foreign keys as incorrect. In this case the LCS correctly classified 15 017 IND's as incorrect foreign keys, while the Dice coefficient correctly classified 15 137 IND's as incorrect foreign keys.

The typical name suffix did not turn out to be a good classifier of correct or incorrect foreign keys. For both the CD shop and the article maintenance experiments the column combinations that were included via an IND's usually all had some sort of typical name suffix thus this property could not be used as a discriminator.

The Earth Movers Distance (EMD) between the value distribution of the parent and child colums quite accurately predicted if an IND was indeed a foreign key. For the calculation of the EMD a threshold of 2 was used as recommended in [76]: The result was that all calculated values were between 0 and 2. For the CD shop example this allowed for a very accurate classification of foreign keys when combined with the LCS. By removing all IND's with an EMD of more than 1 the only IND's that remained were the foreign keys and 3 unclear cases, with no false negatives. Two of the unclear cases could have been modeled as foreign keys, while the other case was a one-on-one link between two tables (for which the foreign key was definied in the reverse direction). For the article maintenance dataset 946 IND's were correctly classified as incorrect foreign keys, although 42 true foreign keys were also falsely classified as incorrect. The combination thereby resulted in 320 remaining IND's of whom 71 were incorrect, 153 were actual foreign keys and 96 were unclear. The last group consisted of IND's for which there was no concensus in the business wether or not a foreign key should be enforced.

Schema extraction evaluation

Though a variety of previous approaches is available for schema extraction the experimental results show that fully automated schema reconstruction is not yet possible.

The most significant problem was scalability. Since most related work was tested on datasets that were at least a factor 10 smaller this could not be predicted beforehand. A notable exception here was HCA-Gordian which was tested on datasets that were only a factor 4 smaller – Note that this was the only algorithm that could properly handle the

[12] Due to an unfortunately timed error this statistic is not available

ERP system dataset. It must be noted however that performance for most techniques relied largely on the underlying database, which was PostgreSQL. PostgreSQL was chosen because it was freely available and well documented, but since I have no prior experience with PostgreSQL the performance issues could also (partly) be caused by an ineffective configuration of the system. A few preliminary tests on the CD shop dataset with HSQLDB as an alternative showed that performance was much better in this case, but since HSQLDB does not support database sizes over 16 GB this could not be verified for the larger datasets.

In addition to this overall precision is an issue as well. This is caused by each steps dependence on the result of the previous step. Thus if a step fails to complete than any further steps cannot be executed either. More importantly this also implies that errors multiply: if for example 20% of the primary keys are incorrect than all extracted foreign keys based on these primary keys are also incorrect.

5.2.2 Identify artifact schemas

Experiments with the artifact schema identification approach show that the runtime appears to depend on the size (and possibly distribution) of the data only.

The *column entropies* and *table importance* columns in Table 7 each show part of the step to calculate the importance of a table. The *column entropies* column shows the required time to calculate the entropy for each column, while the *table importance* step shows the required time to calculate the final importance of a table using schema information and the calculated entropies.

The *table distance*, *base clusters* and *cluster expansion* columns each show part of the step to calculate the actual artifact schemas. The *table distance* column shows the required time to calculate the distance between two tables for which a foreign key relation is defined, while the other two columns show the required times for the actual clustering and expansion steps. Note that only these last two steps depend on the number of clusters calculated.

Description	*Column entropies*	*Table importance*	*Table distance*	*Base clusters*	*Cluster expansion*
CD shop (4 clusters)	1.7 s	0.8 s	1.8 s	< 0.1 s	< 0.1 s
Article maintenance (1 cl.)	16h 42 m	51.6 s	4 h 56 m[13]	< 0.1 s	< 0.1 s
Article maintenance (20 cl.)	*identical*	*identical*	*identical*	< 0.1 s	< 0.1 s
Article maintenance (50 cl.)	*identical*	*identical*	*identical*	< 0.1 s	< 0.1 s
Article maintenance (100 cl.)	*identical*	*identical*	*identical*	< 0.2 s	< 0.1 s

Table 7: Artifact schema identification experiment results

The run times in Table 7 show clearly that the majority of the time is required for the steps in which data is processed. Aside from the amount of data these steps only depend linearly on the number of columns or foreign keys in the schema. Therefore the runtime appears to depend on the size of the data only.

For the CD shop dataset the identified artifact schemas correspond exactly with the 3 artifact schemas given in subsection 0 when 4 clusters are identified; the 4th cluster is the *aux* table that is used for the simulation only. When 3 clusters are identified all tables of the *CD* artifact are added to the *quote* artifact, which seems to make little sense. When more than 4 clusters are generated the *cdquote_order* table becomes a separate cluster, which does not make sense either. Because of this and since the actual clustering was quite fast a trial and error approach was feasible to find the correct number of clusters.

[13] This excludes the distance calculation between the largest table and its parent. This calculation was aborted since the run time was over 31 hours.

For the article maintenance dataset no sensible set of artifact schemas could be identified. This was caused by a significant number of missing (composite) foreign keys, meaning that the structure of the database could not be correctly taken into account by the algorithm.

5.3 Artifact lifecycle identification

5.3.1 Create schema to log mapping

This section describes the experimental results with respect to the efficiency of the schema to log mapping techniques; the correctness of the generated mappings is evaluated in section 0 below. Table 8 shows the properties of the event log mappings that were created. Here the *CD*, *Order* and *Quote* log files were generated from the corresponding artifact schema in the CD shop dataset, while the *Superunie mutaties*, *Vib gevaarlijke stof* and *Artikel* event log mappings were generated from the article maintenance dataset.

Description	*Tables*	*Columns*	*Event types*	*List attributes*	*Attribute columns*	*Run time*
CD	3	10	0	0	5	< 0.5 s
Order	6	23	9	2	8	< 0.5 s
Quote	10	35	11	3	13	< 0.5 s
Superunie mutaties	1	195	23	0	171	< 0.1 s
Vib gevaarlijke stof	266	3729	2019	0	3343	2.8 s
Artikel	47	869	127	0	841	1.4 s

Table 8: Mapped event logs properties

For all experiments the execution time was neglible as compared to the execution times of other steps. The total execution time of all steps to contain a schema-to-log mapping was less than 3 seconds for even the large *Vib gevaarlijke stof* artifact. This was as expected, since the running times of the algorithms is at worst polynomial and only meta data is required to construct the mapping. Note that the runtime of the mapping algorithm mostly depends on the number of tables in the artifact schema, although the exact relation is not clear from the experimental results.

5.3.2 Generating traces

The event log mappings created during the previous step were used to generate event logs. Although this was succesfull for the three CD shop artifacts, the generation of the full event log for the *Superunie mutaties* artifact and the *Artikel* artifact did not complete in a reasonable time. As shown in Table 9 the majority of the time was required to generate the event log from the cache database, similar to what was noted for XESame [41].

Description	*Traces*	*Events*	*Attributes*	*Traces to cache*	*Events to cache*	*Cache to event log*
CD	640	0	±7 700	16.5 s	-	23.5 s
Order	119	1 433	±16 300	9.4 s	16.3 s	48.4 s
Quote	219	2 295	± 26 900	18.6 s	15.4 s	61.9 s
Superunie mutaties	324 731	±2.5 M	±75 M	28 h 7 m	6 h 15 m	> 50 h
Artikel	175 209	± 55 M	N/A	20 m	>50 h	N/A
Artikel (1000 traces)	1 000	246 406	1 464 258	8.7 s	1 h 08 m	32 h 10 m

Table 9: Event log generation experiment result

5.3.3 Verifying generated event logs

To measure the precision of the mapping function two experiments were executed with the CD shop dataset using ProM. This involved the verification of the lifecycle of the *quote* and *order* artifacts as captured in their event logs against the existing models available. The event logs were validated using the behavioural conformance plugin[14]. The original lifecycle models that were used were the models available in the "ACSI CD shop (2 artifacts)" plug-in[15]. The exact details of the mapping between the original model and the event logs can be found in Appendix G.

For both the *order* and *quote* the fitness was high (0.99 and 0.95 respectively) for the traces that could be replayed reliably, but the number of traces that could be replayed reliably was quite low (31 out of 119 and 53 out of 219 respectively). For the *order* artifact this was caused by the reorder event, that often occurred first in an order event log, while this was not modeled as such. For the *quote* artifact this was caused by the order in which the *request* and *quote opening* events occurred. The original model assumed this to be the *quote opening* event, but since these events always had identical timestamps this was often the *request* event.

To get an idea of the precision of the mapping function on real-life data the lifecycle of the *Artikel* artifact was generated using an event log with 1 000 traces. The resulting model shown in Figure 27 is highly complex, but this was as expected by domain experts. More detailed analysis showed that several series of activities (such as price updates) shown in the model corresponded well with reality, confirming the accuracy of the approach.

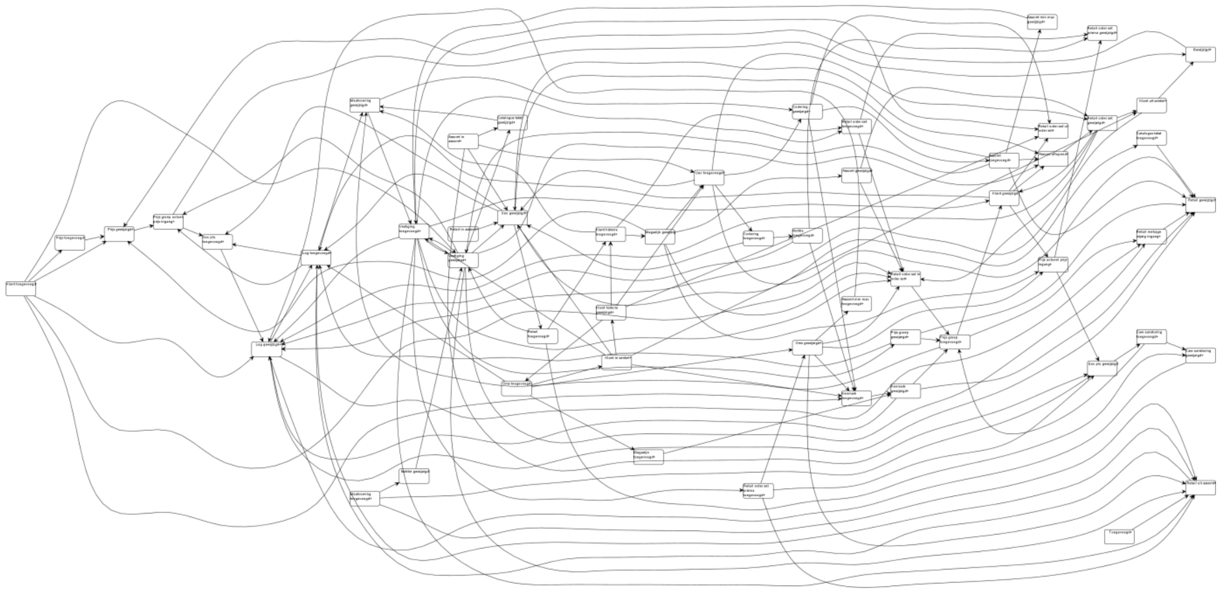


Figure 27: Generated artikel lifecycle

[14] "Replay a log on Petri net for conformance analysis" plug-in created by A. Adriansyah.
[15] Created by B.F. van Dongen. The models were converted to petri nets using the "Construct models for behavioural conformance" plug-in by the same author.

Chapter 6 – Conclusions and future work

6.1 Conclusions

Business process analysis (BPA) in the context of organizations where ERP systems are used can benefit from techniques that make the information in such systems available for BPA. By viewing these organizations as a collection of artifacts many traditional BPA challenges can be solved, but an efficient approach to create an artifact-centric description of an ERP system was missing.

To solve this an approach is presented that decomposes the task of creating an artifact-centric description of an ERP system into pieces for which support can be provided. For each of these pieces techniques are provided that can aid in finding the relevant information, including a new technique that is to my knowledge the first to aid in the creation of a mapping between a database and an event log. The approach was evaluated empirically showing that the method can be used, but that there are also bounds to several techniques that were presented. Finally it should be noted that manual intervention will likely always remain required, since small errors introduced in early steps tend to be magnified in later steps.

The main contribution of this work is to provide a framework that gives insight into techniques that can be used at any step in the analysis of a business process. This allows a practioner to more easily select techniques that are usefull at any point in time, while for the research community the context of a variety of techniques becomes clearer.

6.2 Future work

6.2.1 Further evaluation of the techniques in each step

Although the aproach in each step was based on careful literature research it was not possible to evaluate each promising technique. In addition to this the experimental evaluation showed that the results of the techniques on domain extraction and artifact schema identification were not as usefull as desired. Therefore a further evaluation of the techniques used at each step would be usefull. A starting point could be the information bottleneck method that was shown to be effective in related work on domain extraction and artifact schema identification.

Futher evaluation of techniques could also pay attention to the efficiency of techniques on larger datasets. Especially in the area of domain extraction many techniques failed on the real life datasets for which they should have been most usefull (i.e. the ERP systems database). Aside from algorithmic aspects of the techniques a further evaluation could also focus on more technical aspects such as the database management system used; results on this could be used to present a set of properties that are required to properly compare published research results.

6.2.2 Global optimizations

Overall efficiency could possibly be improved significantly by re-using metrics calculated in previous steps. It was for example noted that there related work describes both domain extraction and foreign key extraction techniques that use value distribution metrics [35], [47], [73] and that the information bottleneck approach was recommended for both domain extraction and artifact schema identification (i.e. schema summarization) [36], [37], [48], [81], [82]. It could be evaluated what the effect on overall precision en efficiency would be to calculate these metrics once and re-use them in later steps.

6.2.3 Schema extraction when data contains errors

Although a variety of techniques exists for schema extraction an assumption for most of them appears to be that all data is valid. In practice this is not the case though so techniques would need to be developed to handle this. An interesting question here is how many errors are actually acceptable, but a starting point for this could be statistical methods to calculate a confidence interval given an observed number of invalid records and a sample size as described in [56], [104], [105].

6.2.4 Human interaction

This report focussed on automated techniques to efficiëntly create an artifact-centric description of an organization using an ERP system. Due to the large number of techniques required overal precision may be low even if the precision of each separate technique is quite high (e.g. if the precision of each of the 4 artifact schema identification techniques is 0.8 then the overall precision will in general still be only $0.8^4=0.41$). One way to improve this is to introduce a number of evaluation steps where a domain expert evaluates the intermediate results. For this purpose evaluation techniques could be developed and evaluated on the trade-off between the required evaluation time and the change in (overall) precision.

6.2.5 Analyzing interactions between artifacts

The full behaviour of artifacts is described by the combination of their inner lifecycle and their interaction with other artifacts, but no process discovery approach exists that generates the interaction behaviour from event logs. A few starting points were identified that could be used to discover such behaviour:

- An event can be uniquely identified by its event column and eventID between different event logs. Thus one can identify if the exact same event occurred in multiple event logs, indicating that their must have been interaction at that point. This holds true even if additional conditions are added to an event mapping manually: These conditions may prevent an event from being generated, but it will still be identified by the same eventID if it is generated.
- The columns identifying another artifact instance are often included as instance or event attributes. Since the mappings defined in this report specify which columns are included it should be relatively straightforward to point out when this occurs. This information could then be included more explicitly in the event log using some yet-to-be-defined notation.

6.2.6 Dropping assumptions on the data

The method could be extended to be used with unstructured data by prepending an information extraction step. Information extraction is defined here as "extracting structure (e.g. tables) from unstructured data (e.g. text)" [32]. It takes as input an unstructured or semistructured dataset and produces a set of structured (i.e. tabular) data. A variety of information extraction techniques has been developed over the last two decades; a starting point for this would be [33], [34].

Bibliography

[1] R. K. L. Ko, S. S. G. Lee, and E. W. Lee, "Business Process Management (BPM) Standards: a Survey," *Business Process Management Journal*, vol. 15, no. 5, pp. 744–791, Sep. 2009.

[2] W. M. P. van der Aalst, A. ter Hofstede, and M. Weske, "Business Process Management: A Survey," in *Business Process Management*, vol. 2678, Springer Berlin / Heidelberg, 2003, p. 1019–1019.

[3] C. Pedrinaci, J. Domingue, and A. Alves de Medeiros, "A Core Ontology for Business Process Analysis," in *The Semantic Web: Research and Applications*, vol. 5021, S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, Eds. Springer Berlin / Heidelberg, 2008, pp. 49–64.

[4] M. Hammer and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution*. Harper Business, 1993.

[5] P. Trkman, "The Critical Success Factors of Business Process Management," *International Journal of Information Management*, vol. 30, no. 2, pp. 125–134, Apr. 2010.

[6] M. Chinosi and A. Trombetta, "BPMN: An Introduction to the Standard," *Computer Standards & Interfaces*, vol. 34, no. 1, pp. 124–134, Jan. 2012.

[7] W. M. P. van der Aalst, "The Application of Petri Nets to Workflow Management," *The Journal of Circuits Systems and Computers*, vol. 8, no. 1, pp. 21–66, 1998.

[8] A. Scheer, O. Thomas, and O. Adam, "Process Modeling using Event-Driven Process Chains," in *Process-Aware Information Systems: Bridging People and Software through Process Technology*, M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede, Eds. Hoboken, NJ, USA: John Wiley & Sons, 2005, pp. 119–145.

[9] W. Aalst, A. Adriansyah, A. K. A. Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. Brand, R. Brandtjen, J. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. Leoni, P. Delias, B. F. Dongen, M. Dumas, S. Dustdar, D. Fahland, D. R. Ferreira, W. Gaaloul, F. Geffen, S. Goel, C. Günther, A. Guzzo, P. Harmon, A. Hofstede, J. Hoogland, J. E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. Rosa, F. Maggi, D. Malerba, R. S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H. R. Motahari-Nezhad, M. Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H. Seguel Pérez, R. Seguel Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, and M. Wynn, "Process Mining Manifesto," in *Business Process Management Workshops*, vol. 99, F. Daniel, K. Barkaoui, and S. Dustdar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 169–194.

[10] J. F. Cox and J. H. Blackstone, Eds., *APICS Dictionary*, 11th ed. Amer Production & Inventory, 2004.

[11] Fenella Scott and Jim Shepherd, "The Steady Stream of ERP Investments," AMR Research, Market research G00187189, Aug. 2002.

[12] METAGroup, "The state of ERP services," META Group, Stamford, CT, Market research, 2004.

[13] C. Cartman and A. Salazar, "The Influence of Organisational Size, Internal IT Capabilities, and Competitive and Vendor Pressures on ERP Adoption in SMEs," *International Journal of Enterprise Information Systems*, vol. 7, no. 3, pp. 68–92, 33 2011.

[14] J. Cardoso, R. P. Bostrom, and A. Sheth, "Workflow Management Systems and ERP Systems: Differences, Commonalities, and Applications," *Information Technology and Management*, vol. 5, pp. 319–338, 2004.

[15] D. A. M. Piessens, "Event Log Extraction from SAP ECC 6.0," Eindhoven University of Technology, Eindhoven, The Netherlands, Master's thesis, Apr. 2011.

[16] M. van Giessel, "Process Mining in SAP R/3: A Method for Applying Process Mining to SAP R/3," Eindhoven University of Technology, Eindhoven, The Netherlands, 2004.

[17] A. Ramesh, "Process mining in PeopleSoft," Eindhoven University of Technology, Eindhoven, The Netherlands, Master's thesis, 2006.

[18] W. M. P. van der Aalst, M. Weske, and D. Grünbauer, "Case Handling: A New Paradigm for Business Process Support," *Data & Knowledge Engineering*, vol. 53, no. 2, pp. 129–162, May 2005.

[19] P. Soffer, B. Golany, and D. Dori, "ERP Modeling: a Comprehensive Approach," *Information Systems*, vol. 28, no. 6, pp. 673–690, Sep. 2003.

[20] Chia-Chia Lin and Dong-Her Shih, "Information System Reengineering for Enterprise Resource Planning as Businesses Adapting to the E-business Era," in *WRI World Congress on Software Engineering, 2009. WCSE '09*, 2009, vol. 3, pp. 222–226.

[21] J. E. Ingvaldsen and J. A. Gulla, "Preprocessing support for large scale process mining of SAP transactions," in *Proceedings of the 2007 international conference on Business process management*, Berlin, Heidelberg, 2008, pp. 30–41.

[22] I. E. A. Segers, "Deloitte Enterprise Risk Services: Investigating the Application of Process Mining for Auditing Purposes," Eindhoven University of Technology, Eindhoven, The Netherlands, Master's thesis, 2007.

[23] A. Khan, A. Lodhi, V. Köppen, G. Kassem, and G. Saake, "Applying process mining in SOA environments," in *Proceedings of the 2009 international conference on Service-oriented computing*, Berlin, Heidelberg, 2009, pp. 293–302.

[24] W. M. P. van der Aalst, P. Barthelmess, C. A. Ellis, and J. Wainer, "Proclets: A Framework for Lightweight Interacting Workflow Processes," *International Journal of Cooperative Information Systems*, vol. 10, no. 4, pp. 443–481, 2001.

[25] T. Heath, "Siena: a Tool for Modeling and Executing Artifact-Centric Business Processes," Università di Roma "La Sapienza," 15-Dec-2009.

[26] M. Dumas, "On the Convergence of Data and Process Engineering," in *Advances in Databases and Information Systems*, vol. 6909, J. Eder, M. Bielikova, and A. M. Tjoa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 19–26.

[27] R. Liu, K. Bhattacharya, and F. Wu, "Modeling Business Contexture and Behavior Using Business Artifacts," in *Advanced Information Systems Engineering*, vol. 4495, Springer Berlin / Heidelberg, 2007, pp. 324–339.

[28] A. Nigam and N. S. Caswell, "Business Artifacts: An Approach to Operational Specification," *IBM Systems Journal*, vol. 42, no. 3, pp. 428 – 445, 2003.

[29] K. Bhattacharya, R. Guttman, K. Lyman, I. I. I. Heath, S. Kumaran, P. Nandi, F. Wu, P. Athma, C. Freiberg, L. Johannsen, and A. Staudt, "A Model-Driven Approach to Industrializing Discovery Processes in Pharmaceutical Research," *IBM Systems Journal*, vol. 44, no. 1, pp. 145–162, 2005.

[30] T. M. Connolly and C. E. Begg, *Database Systems: A Practical Approach to Design, Implementation and Management*, 5th ed. Addison Wesley, 2009.

[31] A. Silberschatz, H. Korth, and S. Sudarshan, *Database System Concepts*, 5th ed. McGraw-Hill Science/Engineering/Math, 2005.

[32] D. Srivastava, “Schema extraction (presentation),” New York, NY, USA, 2010.

[33] S. Sarawagi, “Information Extraction,” *Found. Trends databases*, vol. 1, pp. 261–377, Mar. 2008.

[34] J. Turmo, A. Ageno, and N. Català, “Adaptive information extraction,” *ACM Comput. Surv.*, vol. 38, Jul. 2006.

[35] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava, “On Multi-Column Foreign Key Discovery,” *Proc. VLDB Endow.*, vol. 3, pp. 805–814, Sep. 2010.

[36] D. Srivastava, “Schema Extraction,” in *Proceedings of the 19th ACM international conference on Information and knowledge management*, New York, NY, USA, 2010, pp. 3–4.

[37] B. Ahmadi, M. Hadjieleftheriou, T. Seidl, D. Srivastava, and S. Venkatasubramanian, “Type-Based Categorization of Relational Attributes,” in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, New York, NY, USA, 2009, pp. 84–95.

[38] R. J. Miller and P. Andritsos, “On Schema Discovery,” *IEEE Data Engineering Bulletin*, vol. 26, no. 3, pp. 39–44, 2003.

[39] C. Yu and H. V. Jagadish, “Schema Summarization,” in *Proceedings of the 32nd international conference on Very large data bases*, 2006, pp. 319–330.

[40] S. Fortunato, “Community Detection in Graphs,” *Physics Reports*, vol. 486, no. 3–5, pp. 75–174, Feb. 2010.

[41] J. C. A. M. Buijs, “Mapping Data Sources to XES in a Generic Way,” Eindhoven University of Technology, Eindhoven, The Netherlands, Master’s thesis, 2010.

[42] S. Dustdar, T. Hoffmann, and W. M. P. van der Aalst, “Mining of ad-hoc business processes with TeamLog,” *Data Knowl. Eng.*, vol. 55, pp. 129–158, Nov. 2005.

[43] A. Rozinat, S. Zickler, M. Veloso, W. M. P. van der Aalst, and C. McMillen, “Analyzing Multi-agent Activity Logs Using Process Mining Techniques,” *Distributed Autonomous Robotic System*, no. 8, p. 251, 2009.

[44] W. M. P. van der Aalst, “Process mining in CSCW systems,” in *International Conference on Computer Supported Cooperative Work in Design*, Los Alamitos, CA, USA, 2005, vol. 1, pp. 1–8 Vol. 1.

[45] W. Poncin, “Process Mining Software Repositories,” Eindhoven University of Technology, Eindhoven, The Netherlands, Master’s thesis, 2010.

[46] C. W. Günther, “XES - Standard Definition,” presented at the Event London, 2009.

[47] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava, “Automatic discovery of attributes in relational databases,” in *Proceedings of the 2011 international conference on Management of data*, New York, NY, USA, 2011, pp. 109–120.

[48] P. Andritsos, R. J. Miller, and P. Tsaparas, “Information-Theoretic Tools for Mining Database Structure from Large Data Sets,” in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 2004, pp. 731–742.

[49] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk, “Mining Database Structure; or, How to Build a Data Quality Browser,” in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, 2002, p. 240.

[50] N. Slonim and N. Tishby, "Document Clustering using Word Clusters via the Information Bottleneck Method," in *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 2000, pp. 208–215.

[51] N. Slonim and N. Tishby, "Agglomerative Information Bottleneck," in *NIPS*, Denver, Colorado, USA, 1999, vol. 12, pp. 617–623.

[52] Y. Ren, Y. Ye, and G. Li, "The Density-Based Agglomerative Information Bottleneck," in *PRICAI 2008: Trends in Artificial Intelligence*, vol. 5351, T.-B. Ho and Z.-H. Zhou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 333–344.

[53] P. Andritsos, P. Tsaparas, R. J. Miller, and K. C. Sevcik, "LIMBO: Scalable Clustering of Categorical Data," in *Advances in Database Technology - EDBT 2004*, vol. 2992, E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm, and E. Ferrari, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 123–146.

[54] M. Ester, H. Kriegel, J. S, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," p. 226–231, 1996.

[55] J. Kivinen and H. Mannila, "Approximate Inference of Functional Dependencies from Relations," in *Selected papers of the fourth international conference on Database theory*, Amsterdam, The Netherlands, The Netherlands, 1995, pp. 129–149.

[56] K. Krishnamoorthy and J. Peng, "Some Properties of the Exact and Score Methods for Binomial Proportion and Sample Size Calculation," *Communications in Statistics: Simulation and Computation*, vol. 36, no. 6, pp. 1171–1186, 2007.

[57] H. Mannila and H. Toivonen, "Levelwise Search and Borders of Theories in Knowledge Discovery," *Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 241–258, 1997.

[58] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma, "Discovering all Most Specific Sentences," *ACM Trans. Database Syst.*, vol. 28, no. 2, pp. 140–174, Jun. 2003.

[59] Y. Sismanis, P. Brown, P. J. Haas, and B. Reinwald, "GORDIAN: Efficient and Scalable Discovery of Composite Keys," in *Proceedings of the 32nd international conference on Very large data bases*, 2006, pp. 691–702.

[60] Z. Abedjan and F. Naumann, "Advancing the Discovery of Unique Column Combinations," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, New York, NY, USA, 2011, pp. 1565–1570.

[61] Z. Abedjan and F. Naumann, "Heft 51: Advancing the Discovery of Unique Column Combinations," Hasso-Plattner-Institute, Potsdam, Technical report Heft 51, 2011.

[62] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies," 1999.

[63] M. Kryszkiewicz and P. Lasek, "FUN: Fast Discovery of Minimal Sets of Attributes Functionally Determining a Decision Attribute," in *Transactions on Rough Sets IX*, vol. 5390, J. F. Peters, A. Skowron, and H. Rybiński, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 76–95.

[64] H. Yao and H. J. Hamilton, "Mining Functional Dependencies from Data," *Data Mining and Knowledge Discovery*, vol. 16, pp. 197–219, Sep. 2007.

[65] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga, "CORDS : Automatic Discovery of Correlations and Soft Functional Dependencies," *Proceedings of the*

2004 ACM SIGMOD international conference on Management of data, pp. 647–658, 2004.

[66] M. Kantola, H. Mannila, K. Räihä, and H. Siirtola, "Discovering Functional and Inclusion Dependencies in Relational Databases," *International Journal of Intelligent Systems*, vol. 7, no. 7, pp. 591–607, Sep. 1992.

[67] F. De Marchi, S. Lopes, and J.-M. Petit, "Efficient Algorithms for Mining Inclusion Dependencies," in *Advances in Database Technology — EDBT 2002*, vol. 2287, Springer Berlin / Heidelberg, 2002, pp. 199–214.

[68] F. D. Marchi, S. Lopes, and J.-M. Petit, "Unary and n-Ary Inclusion Dependency Discovery in Relational Databases," *J. Intell. Inf. Syst.*, vol. 32, no. 1, pp. 53–73, Feb. 2009.

[69] J. Bauckmann, U. Leser, F. Naumann, and V. Tietz, "Efficiently Detecting Inclusion Dependencies," in *IEEE 23rd International Conference on Data Engineering, 2007. ICDE 2007*, 2007, pp. 1448–1450.

[70] J. Bauckmann, U. Leser, and F. Naumann, "Efficient and Exact Computation of Inclusion Dependencies for Data Integration," Hasso-Plattner-Institute, Potsdam, Technical report Heft 34, 2010.

[71] F. D. Marchi and J.-M. Petit, "Zigzag: a New Algorithm for Mining Large Inclusion Dependencies in Databases," in *Proceedings of the Third IEEE International Conference on Data Mining*, Washington, DC, USA, 2003, p. 27–.

[72] A. Koeller and E. A. Rundensteiner, "Discovery of High-Dimensional Inclusion Dependencies," in *19th International Conference on Data Engineering, 2003. Proceedings*, 2003, pp. 683– 685.

[73] A. Koeller and E. Rundensteiner, "Heuristic Strategies for the Discovery of Inclusion Dependencies and Other Patterns," in *Journal on Data Semantics V*, vol. 3870, S. Spaccapietra, P. Atzeni, W. Chu, T. Catarci, and K. Sycara, Eds. Springer Berlin / Heidelberg, 2006, pp. 185–210.

[74] A. Rostin, O. Albrecht, J. Bauckmann, F. Naumann, and U. Leser, "A Machine Learning Approach to Foreign Key Discovery," presented at the 12th International Workshop on the Web and Databases (WebDB 2009), 2009.

[75] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, 1st ed. Cambridge University Press, 2008.

[76] O. Pele and M. Werman, "Fast and Robust Earth Mover's Distances," in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 460–467.

[77] L. Bergroth, H. Hakonen, and T. Raita, "A Survey of Longest Common Subsequence Algorithms," in *Seventh International Symposium on String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings*, 2000, pp. 39–48.

[78] G. Kondrak, D. Marcu, and K. Knight, "Cognates Can Improve Statistical Translation Models," *In Proceedings of HLT-NAACL 2003 (Companion Volume)*, p. 46–48, 2003.

[79] T. J. Teorey, G. Wei, D. L. Bolton, and J. A. Koenig, "ER Model Clustering as an Aid for User Communication and Documentation in Database Design," *Commun. ACM*, vol. 32, no. 8, pp. 975–987, Aug. 1989.

[80] T. M. Cover and J. A. Thomas, "Entropy, Relative Entropy and Mutual Information," in *Elements of Information Theory*, 2nd ed., Wiley-Interscience, 2006, pp. 12–41.

[81] X. Yang, C. M. Procopiuc, and D. Srivastava, "Summarizing relational databases," *Proc. VLDB Endow.*, vol. 2, pp. 634–645, Aug. 2009.

[82] X. Yang, C. M. Procopiuc, and D. Srivastava, "Summary Graphs for Relational Database Schemas," in *Proceedings of the VLDB Endowment*, Seatle, Washington, 2011, vol. 4.

[83] W. Wu, B. Reinwald, Y. Sismanis, and R. Manjrekar, "Discovering Topical Structures of Databases," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 2008, pp. 1019–1030.

[84] N. Gehrke, "Basic Principles of Financial Process Mining A Journey through Financial Data in Accounting Information Systems," *AMCIS 2010 Proceedings*, Aug. 2010.

[85] V. Rubin, C. W. Günther, W. M. P. van der Aalst, E. Kindler, B. F. Van Dongen, and W. Schäfer, "Process mining framework for software processes," in *Proceedings of the 2007 international conference on Software process*, Berlin, Heidelberg, 2007, pp. 169–181.

[86] K. van Uden, "Extracting user profiles with Process Mining at Philips Medical Systems," Eindhoven University of Technology, Eindhoven, The Netherlands, Master's thesis, 2008.

[87] C. Günther and W. van der Aalst, "A Generic Import Framework for Process Event Logs," in *Business Process Management Workshops*, vol. 4103, Springer Berlin / Heidelberg, 2006, pp. 81–92.

[88] H. M. W. Verbeek, J. C. A. M. Buijs, B. F. Dongen, and W. M. P. van der Aalst, "XES, XESame, and ProM 6," in *Information Systems Evolution*, vol. 72, W. Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, P. Soffer, and E. Proper, Eds. Springer Berlin Heidelberg, 2011, pp. 60–75.

[89] Günther, "Process mining in flexible environments," PhD thesis, Technische Universiteit Eindhoven, 2009.

[90] A. de Medeiros, C. Pedrinaci, W. van der Aalst, J. Domingue, M. Song, A. Rozinat, B. Norton, and L. Cabral, "An Outlook on Semantic Business Process Mining and Monitoring," in *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, vol. 4806, R. Meersman, Z. Tari, and P. Herrero, Eds. Springer Berlin / Heidelberg, 2007, pp. 1244–1255.

[91] B. F. Van Dongen, "Process Mining: Fuzzy Clustering and Performance Visualization," in *BPM Workshops*, 2009, pp. 158–169.

[92] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[93] D. Fahland, M. Leoni, B. F. Dongen, and W. M. P. Aalst, "Behavioral Conformance of Artifact-Centric Process Models," in *Business Information Systems*, vol. 87, W. Abramowicz, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 37–49.

[94] J. E. Cook and A. L. Wolf, "Automating Process Discovery through Event-Data Analysis," in *Proceedings of the 17th international conference on Software engineering*, New York, NY, USA, 1995, pp. 73–82.

[95] A. Tiwari, C. J. Turner, and B. Majeed, "A Review of Business Process Mining: State-of-the-Art and Future Trends," *Business Process Management Journal*, vol. 14, no. 1, pp. 5–22, Feb. 2008.

[96] C. Günther and W. van der Aalst, "Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics," in *Business Process Management*, vol. 4714, Springer Berlin / Heidelberg, 2007, pp. 328–343.

[97] A. de Medeiros, A. Weijters, and W. van der Aalst, "Genetic Process Mining: an Experimental Evaluation," *Data Mining and Knowledge Discovery*, vol. 14, no. 2, pp. 245–304, Apr. 2007.

[98] W. van der Aalst, A. de Medeiros, and A. Weijters, "Genetic Process Mining," in *Applications and Theory of Petri Nets 2005*, vol. 3536, Springer Berlin / Heidelberg, 2005, p. 985.

[99] C. Bratosin, N. Sidorova, and W. van der Aalst, "Distributed Genetic Process Mining," in *2010 IEEE Congress on Evolutionary Computation (CEC)*, 2010, pp. 1–8.

[100] M. Song, C. W. Günther, and W. M. P. Aalst, "Trace Clustering in Process Mining," in *Business Process Management Workshops*, vol. 17, Springer Berlin Heidelberg, 2009, pp. 109–120.

[101] R. Jagadeesh Chandra Bose and W. van der Aalst, "Trace Alignment in Process Mining: Opportunities for Process Diagnostics," in *Business Process Management*, vol. 6336, Springer Berlin / Heidelberg, 2010, pp. 227–242.

[102] M. Bozkaya, J. Gabriels, and J. Werf, "Process Diagnostics: A Method Based on Process Mining," in *International Conference on Information, Process, and Knowledge Management, 2009. eKNOW '09*, 2009, pp. 22–27.

[103] D. Fahland, M. Leoni, B. F. Dongen, and W. M. P. Aalst, "Conformance Checking of Interacting Processes with Overlapping Instances," in *Business Process Management*, vol. 6896, S. Rinderle-Ma, F. Toumani, and K. Wolf, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 345–361.

[104] T. D. Ross, "Accurate Confidence Intervals for Binomial Proportion and Poisson Rate Estimation," *Computers in Biology and Medicine*, vol. 33, no. 6, pp. 509 – 531, 2003.

[105] A. Agresti and B. A. Coull, "Approximate Is Better than 'Exact' for Interval Estimation of Binomial Proportions," *The American Statistician*, vol. 52, no. 2, pp. 119–126, May 1998.

[106] P. A. Smart, H. Maddern, and R. S. Maull, "Understanding Business Process Management: Implications for Theory and Practice," *British Journal of Management*, vol. 20, no. 4, pp. 491–507, Dec. 2009.

[107] R. G. Lee and B. G. Dale, "Business Process Management: a Review and Evaluation," *Business Process Management Journal*, vol. 4, no. 3, pp. 214–225, 1998.

[108] D. J. Elzinga, T. Horak, Chung-Yee Lee, and C. Bruner, "Business Process Management: Survey and Methodology," *Engineering Management, IEEE Transactions on*, vol. 42, no. 2, pp. 119–128, 1995.

[109] J. A. Ward, "Continuous Process Improvement," *Information Systems Management*, vol. 11, no. 2, pp. 74–76, 1994.

[110] H. J. Harrington, *Business Process Improvement: the Breakthrough Strategy for Total Quality, Productivity, and Competitiveness*. McGraw-Hill Professional, 1991.

[111] R. Andersson, H. Eriksson, and H. Torstensson, "Similarities and Differences between TQM, Six Sigma and Lean," *The TQM Magazine*, vol. 18, no. 3, pp. 282–296, May 2006.

[112] L. Donaldson, *The Contingency Theory of Organizations*. SAGE, 2001.

[113] L. Donaldson, "Strategy and Structural Adjustment to Regain Fit and Performance: In Defence of Contingency Theory," *Journal of Management Studies*, vol. 24, no. 1, pp. 1–24, Jan. 1987.

[114] H. Mintzberg, *Mintzberg on management: Inside our strange world of organizations*. New York, NY, USA: Free Press, 1989.

[115] H. Mintzberg, "Structure in 5's: A Synthesis of the Research on Organization Design," *Management Science*, vol. 26, no. 3, pp. 322–341, Mar. 1980.

[116] N. A. Morton and Q. Hu, "Implications of the Fit Between Organizational Structure and ERP: A Structural Contingency Theory Perspective," *International Journal of Information Management*, vol. 28, no. 5, pp. 391–402, Oct. 2008.

[117] J. J. Korhonen., "On the Lookout for Organizational Effectiveness – Requisite Control Structure in BPM Governance," presented at the 1st International Workshop on BPM Governance 2007 - WoGo 2007, Brisbane, Australia, 2007.

[118] M. Al-Mashari and M. Zairi, "BPR Implementation Process: an Analysis of Key Success and Failure Factors," *Business Process Management Journal*, vol. 5, no. 1, pp. 87–112, 1999.

[119] J. B. Hill, J. Sinur, D. Flint, and M. J. Melenovsky, "Gartner's Position on Business Process Management, 2006," Gartner, Inc., Stamford, CT, USA, G00136533, Feb. 2006.

[120] C. C. H. Law and E. W. T. Ngai, "ERP Systems Adoption: An Exploratory Study of the Organizational Factors and Impacts of ERP Success," *Information & Management*, vol. 44, no. 4, pp. 418–432, Jun. 2007.

Table of tables

Table 1: CD shop metadata 9
Table 2: Log mapping symbols 17
Table 3: Dataset statistics 45
Table 4: Domain extraction experiment statistics 46
Table 5: Primary key experiment statistics 47
Table 6: Foreign key experiment statistics 48
Table 7: Artifact schema identification experiment results 49
Table 8: Mapped event logs properties 50
Table 9: Event log generation experiment result 50
Table 10: Organizational types 66
Table 11: Notations used 68
Table 12: ACSI to XESame translation 71

Table of formulas

Formula 1: Sample size 22
Formula 2: DirectParents 34
Formula 3: AllParents 34
Formula 4: DirectChildren 34
Formula 5: AllChildren 34
Formula 6: $T_{\text{InstanceEvents}}$ 75

Table of figures

Figure 1: Process support systems (adapted from [18]) 2
Figure 2: Traditional view on processes 4
Figure 3: Artifact view on business processes.. 5
Figure 4: General idea of the approach... 6
Figure 5: CD shop database schema 8
Figure 6: Quote schema 10
Figure 7: Order schema..11
Figure 8: Quote and order lifecycles...11
Figure 9: CD schema 12
Figure 10: Overall method 13
Figure 11: Class diagram of mapping domain model 16
Figure 12: Artifact schema identification approach 19
Figure 13: Schema expansion example 30
Figure 14: Artifact lifecycle identification approach 31
Figure 15: CreateTraceMapping algorithm 33
Figure 16: AllParents algorithm 34
Figure 17: AllChildren algorithm 34
Figure 18: IdentifyEventTypes algorithm 35
Figure 19: SelectParentsWithoutEvents algorithm 36
Figure 20: SelectInstanceChildTables algorithm..36
Figure 21: SelectEventChildTables algorithm..37
Figure 22: CreateMapping algorithm 38
Figure 23: SplitOneAndMany algorithm 39
Figure 24: Cache database 42
Figure 25: Discovered Quote lifecycle...43
Figure 26: Discovered Order lifecycle 43
Figure 27: Generated artikel lifecycle 51
Figure 28: Class diagram of XESame domain model (from [41])...............................69
Figure 29: Prototype architecture 83
Figure 30: HSQLDB connection parameters...84
Figure 31: Internal storage mechanisms...84
Figure 32: Prototype trace information 85
Figure 33: Prototype trace settings 85
Figure 34: General domain extraction parameters dialog 89
Figure 35: Example domain clustering method parameters dialog..............................90
Figure 36: Select tables dialog 91
Figure 37: Domain clustering results screen..91
Figure 38: Primary key extraction parameters dialog 92
Figure 39: Primary key extraction results screen 93
Figure 40: Foreign key extraction parameters dialog...93
Figure 41: Foreign keys extraction results screen (left) 94
Figure 42: Foreign keys extraction results screen (right) 94
Figure 43: Cluster tables parameters dialog 95
Figure 44: Artifact identification results screen 95
Figure 45: Create artifact parameters dialog 96
Figure 46: Attribute & event selection parameters dialog 96
Figure 47: Event type selection and attribute assignment results screen.....................97
Figure 48: Create event log mapping parameters dialog 98
Figure 49: Create event log from datasource parameters dialog 98
Figure 50: Create event log from cache parameters dialog 99
Figure 51: Order original model - log mapping 101
Figure 52: Quote original model - log mapping...102

Appendix A: Why process mining of ERP systems?

Both BPM and ERP systems seem to be most usefull in stable environments, since both rely on standardization to achieve a variety of benefits. BPM focusses on the standardization of processes, while ERP systems focus on the standardization of data and activities. In addition to this ERP systems store a lot of data elektronically. Since process mining uses elektronic data to execute the analysis of BPM it seems to be a natural fit for organizations that have an ERP system in place. This results in the following hypothesis:

Organizations that have an ERP system in place can benefit from process mining.

In the following sections previous research results will be shown that support the hypothesis stated above. The focus will be on organizational contingency theory and succes factors for ERP and process mining.

A.I Business Process Management

Business Process Management (BPM) has received a large amount of attention since the introduction of business process re-engineering [106]. A large variety of definitions exist for BPM, focussing on concepts such as process-centric, customer-centric, systematic and/or continuous approaches and activities such as analysis, improvement and management [2], [5], [106–108]. A general definition would be all business process-centric approaches to systematically analyze and continuously improve activities within an organization.

Aside from the term Business Process Management, several other terms exist to describe management practices aimed at improving business processes. Examples are Continuous Process Improvement (CPI, [109]), Business Process Improvement (BPI, [110]), Total Quality Management (TQM), Lean and Six Sigma [111]. Although there are differences between these approaches all of them are implied by BPM as defined here.

A.II Organizational context

A.II.I Contingency theory

Organizational contingency theory states that organizational characteristics (such as structure) follow contingencies (internal and environmental factors such as organization size and the rate of technological change) [112]. The key idea is that organizational effectiveness depends on the fit of the organization with these contingencies, that organizations want to be as effective as possible and thus organizations tend to adapt themselves to fit [5], [113], [114]. For both ERP systems and BPM research has been done based on contingency theory; i.e. what circumstances lead to successfull application of these in organizations.

A.II.II Types of organizations

In [115] and later [114] Mintzberg presents seven pure configurations of organizations with several distinctive characteristics. As per the contingency principle each of the configurations is most effective depending on a given set of contingencies [114], [116]. Table 10 shows a brief overview of these configurations.

Pure type	*Characteristics*	*Pure type*	*Characteristics*
Entre-preneurial	- Simple, informal, flexible - Young or in crisis/turnaround	Missionary	- Clear, focused, inspiring, distinctive mission - Small, loosely organized units
Machine	- Centralized, highly formalized and efficiënt - Larger, more mature organization	Innovative	- Organic, selectively decentralized - Innovative but inefficiënt

Pure type	*Characteristics*	*Pure type*	*Characteristics*
Diversified	- Loosely coupled divisions under centralized administrative headquarters - Divisions tend to be machine form	Professional	- Decentralized but bureaucratic - Highly skilled operating professionals who value autonomy
Political	- Conventional notions of coördination absent, replaced by informal power (politics)		- Serves to bring change blocked by legitimate systems of influence - Unstable, unless supported by environment

Table 10: Organizational types

These configurations can be used to describe organizations as aspects of the different pure types [114]. In this way an organization is described by how parts of and/or forces within the organization resembles the different pure types.

A.II.III Suitability of BPM and ERP for organizational types

The basis of BPM is that processes are designed instead of evolving naturally, which makes BPM best applicable in a formal top-down organization (the machine type). This is confirmed by the survey results in [106] which show that architecture and measurement are mentioned most often in an interview about BPM (besides conceptual components such as "process"). In addition a variety of papers mention the importance of control and measurement for BPM [5], [107], [108], [117–119], which also indicates a formal, machine-like structure as the best fit for BPM.

Like BPM, ERP systems are best applicable for formal top-down organizations (the machine type) as well [116]. Thus the same type of organizations can benefit from both BPM and ERP systems.

A.III Success factors

For process mining to be succesful event data of sufficient quality is required. The Process Mining Manifesto states that the minimum requirement is that events are automatically recorded and that there should be some sort of guarantee that recorded events match reality [9]. As stated in the manifesto this is the case for ERP systems. An additional quality measure for event logs is completeness: no events should be missing for the process that is being analyzed [9]. ERP systems were not build with event logs in mind, so there is no guarantee that this is the case for these systems, but ERP systems do increase the likelyhood of completeness. Two factors contribute to this:

- ERP systems are meant to aid in cross-departemental activities. The idea is that business processes can be handled in a single system (so there is only one version of the truth).
- ERP systems are data driven by nature, thus information that is used during the process will usually be stored in the ERP system.

As shown in [120] ERP success is higher when business process improvements efforts are performed. Process mining can be used as a tool for business process improvement, therefore making it usefull when ERP systems are in place.

A.IV Process mining of ERP systems creates synergy

As shown in the previous subsections ERP systems and process mining can be used in the same type of organizations. In addition to this both increase eachothers chances of success: the result of using process mining together with ERP systems will be better than using both independent of eachother. Thus organizations that have an ERP system in place will indeed benefit from process mining.

Appendix B: Notations used

Symbol	*Description*
$\mathfrak{S} = (\mathfrak{T}, \mathfrak{F}, \mathfrak{D}, \mathit{column\_domain})$	Schema
$\mathfrak{T} = \{T_1,\ldots,T_n\}$	Set of all tables
$\mathfrak{F} = \{F_1,\ldots,F_n\}$	Set of all foreign keys
$\mathfrak{D} = \{D_1,\ldots,D_n\}$	Set of all domains
$\mathfrak{C} = \{C_1,\ldots C_n\}$	Set of all columns
$\mathit{column\_domain} : \mathfrak{C} \rightarrow \mathfrak{D}$	Function that assigns a domain D to each column C
$T = (\mathbf{C}, \mathbf{C_p})$	Table with columns C and primary key C_p
$\mathbf{C} = \{C_1,\ldots,C_n\}$	Set of columns
$\mathbf{C_p} = \{C_{p1},\ldots,C_{pn}\}$	Primary key consisting of n columns
$F = (T_p,\mathbf{C_p},T_c,\mathbf{C_c})$	Foreign key from parent table T_p with primary key C_p to child table T_c with referencing columns C_c
$\lvert T\rvert$	Number of distinct values in table T
$\#T$	Number of values in table T
$\lvert C\rvert$	Number of distinct values in column C
$\lvert \mathit{Set}\rvert$	Number of (distinct) values in the set
$\mathit{path}(T_1,T_2) = (F_1,\ldots,F_n)$	The shortest path of references from T_1 to T_2
$(C_1,\ldots,C_n) \subseteq (C'_1,\ldots,C'_n)$	Inclusion dependency: $(C_1,\ldots,C_n)$ is pairwise included in $(C'_1,\ldots,C'_n)$
$\mathbf{C} \rightarrow \mathbf{C'}$	Functional dependency: **C** functionally determines **C’**
$\mathfrak{S}_A = (\mathfrak{T}_A, \mathfrak{F}_A, \mathfrak{D}_A, \mathit{column\_domain}, T_m)$	Artifact schema with main table T_m
$A = (\mathfrak{S}_A, \mathbf{ET}, \mathbf{C}_A)$	Artifact with schema $\mathfrak{S}_A$, event types ET and instance attribute columns $\mathbf{C}_A$
$\mathbf{ET} = \{ET_1,\ldots,ET_n\}$	Set of n event types
$ET = (T_{ET}, C_e, \mathbf{C}_a)$	Event type with event table T_{ET}, event column C_e and attribute columns $\mathbf{C}_a$. The event column describes the ordering of the events, most likely containing timestamp values
$\mathbf{C}_A = \{C_{A1}, \ldots C_{An}\}$	Set of n instance attribute columns
$\mathbf{C}_a = \{C_{a1}, \ldots C_{an}\}$	Set of n event type attribute columns
$LM = (\mathrm{name}, TM, \mathbf{EX}, \mathbf{CL}, \mathbf{AG_T}, \mathbf{AG_E})$	Log mapping with artifact name *name*, a trace mapping TM, extensions **EX**, classifiers **CL**, global trace attributes $\mathbf{AG_T}$ and global event attributes $\mathbf{AG_E}$
$GM = (\mathbf{C_{ID}}, T_{From}, \mathbf{F_{Link}}, \mathbf{AM}, \mathbf{LA})$	General mapping item that serves as the basis for a trace mapping, event mapping or list attribute. It consists of ID columns $\mathbf{C_{ID}}$, main table T_{From}, other table links $\mathbf{F_{Link}}$, attribute mappings **AM** and list attributes **LA**. Note that the ID columns correspond to the traceID, eventID and attributeID columns of traces, events and attributes respectively.
$TM = (\mathbf{C}_{TID}, T_{From}, \mathbf{F_{Link}}, \mathbf{EM}, \mathbf{AM_T}, \mathbf{LA_T})$	Trace mapping with traceID columns $\mathbf{C}_{TID}$, main table T_{From}, other table links $\mathbf{F_{Link}}$, event mappings **EM**, attribute mappings $\mathbf{AM_T}$ and list attributes $\mathbf{LA_T}$
$\mathbf{EM} = \{EM_1,\ldots,EM_n\}$	Set of event mappings
$EM = (\mathrm{name}, \mathbf{C}_{EID}, C_e, T_{From}, \mathbf{F_{Link}}, \mathbf{AM_E}, \mathbf{LA_E})$	Event mapping for event *name* with eventID columns $\mathbf{C}_{EID}$ and event column C_e, main table T_{From}, other table links $\mathbf{F_{Link}}$, attribute mappings $\mathbf{AM_E}$ and list attributes $\mathbf{LA_E}$. The event column describes the ordering of the events, most likely containing timestamp values

LA = {LA_1,...,LA_n}	Set of n list attributes
LA = (key, **C**$_{AID}$, T_{From}, **F**$_{\mathbf{Link}}$, **AM**$_{\mathbf{L}}$, **LA**$_{\mathbf{L}}$)	List attribute (an attribute with multiple values) mapping with given *key*, attributeID columns **C**$_{AID}$, main table T_{From}, other table links **F**$_{\mathbf{Link}}$, attribute mappings **AM**$_{\mathbf{L}}$ and list attributes **LA**$_{\mathbf{L}}$
AM = {AM_1,...,AM_n}	Set of n attribute mappings
AM = (key, type, C_a)	Attribute mapping with given *key*, *type* and attribute column C_a
AG = {AT_{G1},...,AT_{Gn}}	Set of n global attributes
AT = (key, type, value)	Attribute with given *key*, *type* and *value*
EX = {EX_1,...,EX_n}	Set of n extensions
EX = (name, prefix, URI)	Extension with given *name*, *prefix* and *URI*
CL = {CL_1,...,CL_n}	Set of n classifiers
CL = (name, keys)	Classifier with given *name* and *keys*

Table 11: Notations used

Appendix C: Translating to XESame

The method decribed in this report provides a way to semi-automatically generate a mapping and event log from a source database to the XES log format, but no way to modify this mapping afterwards. XESame on the other hand provides an easy way to create a custom mapping, but no support to automatically generate (part of) the mapping. The section below describes how the mapping generated by the ACSI method can be translated to a XESame mapping, thus allowing the flexibility of XESame to be combined with the automated support of the ACSI method.

The mapping model of XESame is show in Figure 28 below. The majority of the mapping created by the ACSI method can be translated to this model, but there are some exceptions:

- ListAttributes cannot be translated to the XESame mapping model however, since variable length attribute lists cannot be handled by XESame.
- Trace and event global attributes are generated automatically in XESame based on the available attributes in the mapping. Because of this these elements are not present in the class diagram of XESame and cannot be translated to from the ACSI mapping model.
- The XESame connection element is not present in the ACSI mapping model, so this would need to be added manually.

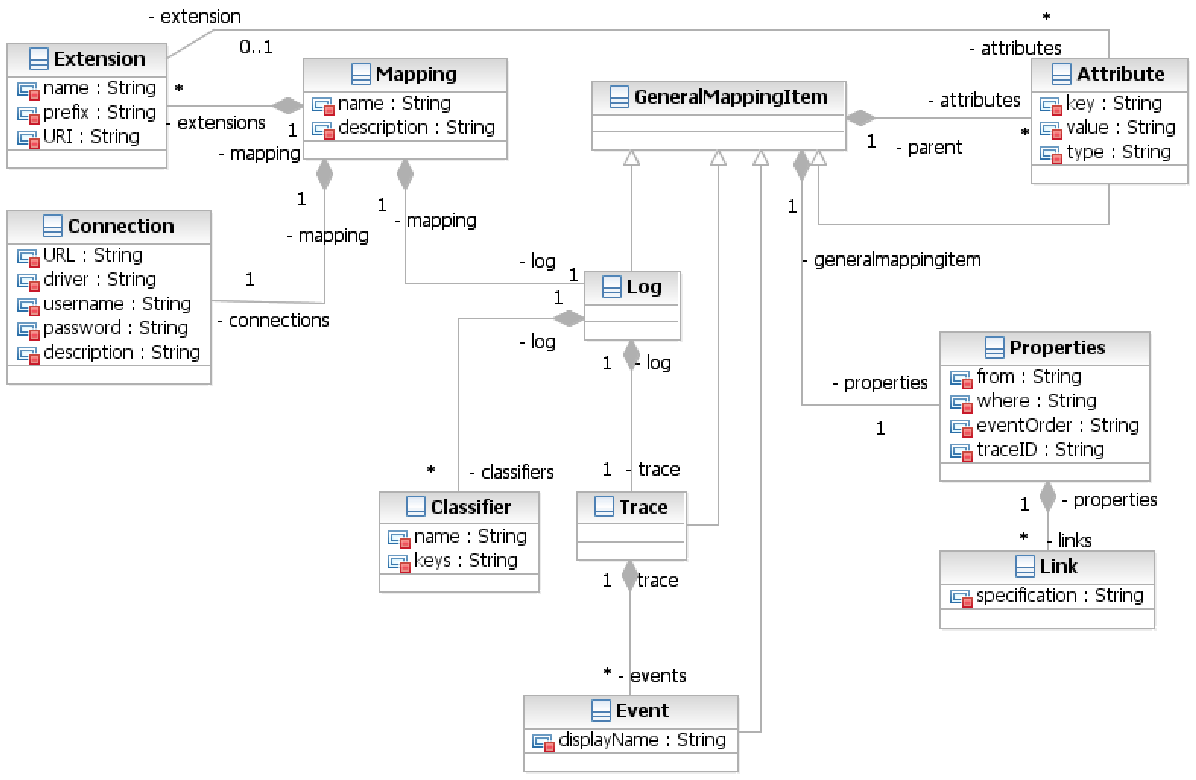


Figure 28: Class diagram of XESame domain model (from [41])

Table 12 below shows how elements in the ACSI mapping model can be translated to elements in the XESame model. Values between square brackets ([]) show the exact values that should be used in the XESame model. Between these brackets *italic* text refers to the value of the named element, while regular text refers to literal values that should be used. Finally, "Identical" means that the two attributes are one-on-one mapping. Therefore no translation is necessay in this case.

ACSI model		*XESame model*		*Additional comments*
Class	*Attribute*	*Class*	*Attribute*	
-	-	Mapping	description	No corresponding element available in ACSI model
Log-Mapping	artifactName	Mapping	name	Mapping name becomes [*artifactName* + ‘ log’]
Extension	name	Extension	name	Identical
	prefix		prefix	Identical
	URI		URI	Identical
Classifier	name	Classifier	name	Identical
	keys		keys	Identical
Trace-Mapping	from	Trace	from	[*Table name*]
	traceID		traceID	All columns joined together, i.e. if the ACSI *traceID* is {A, B} then the XESame *traceID* will be [A + ‘_’ + B][16]
		Attribute (trace)	key value type	Always [traceID] The combined XESame *traceID* The XES type of the *traceID* column or [String] if the number of columns is greater than 1.
		Attribute (trace)	key value type	Always [concept:name] The artifact name followed by the traceID: [*artifactName* + ‘ ’ + *traceID*] Always [String]
		Event	traceID	The combined XESame *traceID*
	link	Link	specification	Join statement created from foreign key, e.g. [*child table name* ON *parent table name*.*parent table primary key column* = *child table name*.*child table column*]
Event-Mapping	from	Event	from	[*Table name*]
	name	Event	displayName	Identical
		Attribute (event)	key value type	Always [concept:name] The ACSI event name [*name*] Always [String]
	eventID	Attribute (event)	key value type	Always [eventID] All columns joined together, i.e. if the ACSI *eventID* is {A, B} then the attribute value will be [A + ‘_’ + B] [16] The XES type of the *eventID* column or [String] if the number of columns is greater than 1
	eventColumn	Event	eventOrder	Identical
		Event	where	[*eventColumn* IS NOT NULL]
		Attribute (event)	key value type	[time:timestamp] if the *eventColumn* has a XES Date type, [time] otherwise [*eventColumn*] The XES type of the *eventColumn*

[16] If "+" is the SQL concatenation operator

ACSI model		*XESame model*		*Additional comments*
Class	*Attribute*	*Class*	*Attribute*	
Event-Mapping	link	Link	specification	Join statement created from foreign key, e.g. [*child table name* ON *parent table name*.*parent table primary key column = child table name*.*child table column*]
Attribute-Mapping	key	Attribute	key	Identical
	type	Attribute	type	Identical
	sourceColumn	Attribute	value	Identical

Table 12: ACSI to XESame translation

Appendix D: More advanced algorithms

D.I Efficiently selecting instance and event tables

The algorithms below show how the selection of instance and event tables can be done more efficiently by keeping track of the validity of tables that are encountered. Here $\mathbf{T}_{\mathbf{Ignore}}$ consists of set of tables that should not be added (again) and should be treated as though they and their parents do not contain event columns. $\mathbf{T}_{\mathbf{Events}}$ consists of the set of tables that should not be added because they or one of their parents contain event columns. Both $\mathbf{T}_{\mathbf{Ignore}}$ and $\mathbf{T}_{\mathbf{Events}}$ can be implemented as bit vectors of length $|\mathcal{T}_{\mathbf{A}}|$, with a value of 1 at the index of a specific table implying membership of the set. Membership of the set can then be calculated in O(1).

In this case each table is checked for event columns once. The next time the table is encountered only the constant time lookup will be executed. When the table is checked it has to evaluate all of its columns $\mathbf{C}$, all of its parent tables $\mathbf{T}_{\mathbf{P}} \subseteq \mathcal{T}_{\mathbf{A}}$ and all of its child tables $\mathbf{T}_{\mathbf{C}} \subseteq \mathcal{T}_{\mathbf{A}}$. It is assumed that a parent table cannot also be a child table. Denote by $|\mathbf{C}|_{max}$ the maximum number of columns in a table. Then to check a table an $O(|\mathbf{C}|+|\mathcal{T}_{\mathbf{A}}|)$ operation is thus required. Therefore the total running time of both *SelectInstanceChildTables* and *SelectEventChildTables* reduces to $O(|\mathcal{T}_{\mathbf{A}}|\cdot(|\mathbf{C}|_{max}+|\mathcal{T}_{\mathbf{A}}|))$.

```
0.  SelectInstanceChildTables($T_0$, $\mathbf{T}_{Ignore}$, $\mathbf{T}_{Events}$, $\mathbb{S}_A$)
1.      $\mathbf{T}_{valid} = \emptyset$
2.      For each $T \in$ DirectChildren($T_0$, $\mathbb{S}_A$)
3.          If $(T \in \mathbf{T}_{Ignore}) \vee (T \in \mathbf{T}_{Events})$ Then
4.              Next T

5.          If (Event columns in T) Then
6.              $\mathbf{T}_{Events} = \mathbf{T}_{Events} \cup \{T\}$
7.          Else
8.              $\mathbf{T}_{Parents}$ = SelectParentsWithoutEvents (T, $\mathbf{T}_{Ignore}$, $\mathbf{T}_{Events}$, $\mathbb{S}_A$)
9.              If Parents with events found for T Then
10.                 $\mathbf{T}_{Events} = \mathbf{T}_{Events} \cup \{T\}$
12.             Else
13.                 $\mathbf{T}_{Ignore} = \mathbf{T}_{Ignore} \cup \{T\} \cup \mathbf{T}_{Parents}$
14.                 $\mathbf{T}_{valid} = \mathbf{T}_{valid} \cup \{T\} \cup \mathbf{T}_{Parents}$
15.                 $\mathbf{T}_{valid} = \mathbf{T}_{valid} \cup$ SelectInstanceChildTables(T, $\mathbf{T}_{Ignore}$, $\mathbf{T}_{Events}$, $\mathbb{S}_A$)
16.     Return $\mathbf{T}_{valid}$

17. $\mathbf{T}_{Instance}$ = SelectInstanceChildTables($T_m$, $\{T_m\}$, $\emptyset$, $\mathbb{S}_A$)
18. $\mathbf{T}_{Instance} = \mathbf{T}_{Instance} \cup$ AllParents ($\mathbf{T}_{Instance}$, $\mathbb{S}_A$)
```

```
0.  $SelectParentsWithoutEvents(\mathrm{T}_0, \mathbf{T}_{\mathbf{Ignore}}, \mathbf{T}_{\mathbf{Events}}, \mathfrak{S}_A)$
1.      $\mathbf{T}_{\mathbf{valid}} = \emptyset$
2.      For each $\mathrm{T} \in DirectParents(\mathrm{T}_0, \mathfrak{S}_A)$
3.          If $(\mathrm{T} \in \mathbf{T}_{\mathbf{Ignore}}) \vee (\mathrm{T} \in \mathbf{T}_{\mathbf{Events}})$ Then
4.              Next T

5.          If (Event columns in T) Then
6.              $\mathbf{T}_{\mathbf{Events}} = \mathbf{T}_{\mathbf{Events}} \cup \{\mathrm{T}\}$
7.          Else
8.              $\mathbf{T}_{\mathbf{Parents}} = SelectParentsWithoutEvents\ (\mathrm{T}, \mathbf{T}_{\mathbf{Ignore}}, \mathbf{T}_{\mathbf{Events}}, \mathfrak{S}_A)$
9.              If Parents with events found for T Then
10.                 $\mathbf{T}_{\mathbf{Events}} = \mathbf{T}_{\mathbf{Events}} \cup \{\mathrm{T}\}$
12.             Else
13.                 $\mathbf{T}_{\mathbf{Ignore}} = \mathbf{T}_{\mathbf{Ignore}} \cup \{\mathrm{T}\} \cup \mathbf{T}_{\mathbf{Parents}}$
14.                 $\mathbf{T}_{\mathbf{valid}} = \mathbf{T}_{\mathbf{valid}} \cup \{\mathrm{T}\} \cup \mathbf{T}_{\mathbf{Parents}}$
15.     Return $\mathbf{T}_{\mathbf{valid}}$
```

```
0.  $SelectEventChildTables(\mathrm{T}_0, \mathbf{T}_{\mathbf{Ignore}}, \mathbf{T}_{\mathbf{Events}}, \mathfrak{S}_A)$
1.      $\mathbf{T}_{\mathbf{valid}} = \emptyset$
2.      For each $\mathrm{T} \in DirectChildren(\mathrm{T}_0, \mathfrak{S}_A)$
3.          If $(\mathrm{T} \in \mathbf{T}_{\mathbf{Ignore}}) \vee (\mathrm{T} \in \mathbf{T}_{\mathbf{Events}})$ Then
4.              Next T

5.          If (Event columns in T) Then
6.              $\mathbf{T}_{\mathbf{Events}} = \mathbf{T}_{\mathbf{Events}} \cup \{\mathrm{T}\}$
7.          Else
8.              $\mathbf{T}_{\mathbf{Parents}} = SelectParentsWithoutEvents\ (\mathrm{T}, \mathbf{T}_{\mathbf{Ignore}}, \mathbf{T}_{\mathbf{Events}}, \mathfrak{S}_A)$
9.              If Parents with events found for T Then
10.                 $\mathbf{T}_{\mathbf{Events}} = \mathbf{T}_{\mathbf{Events}} \cup \{\mathrm{T}\}$
11.             Else
12.                 $\mathbf{T}_{\mathbf{Ignore}} = \mathbf{T}_{\mathbf{Ignore}} \cup \{\mathrm{T}\} \cup \mathbf{T}_{\mathbf{Parents}}$
13.                 $\mathbf{T}_{\mathbf{valid}} = \mathbf{T}_{\mathbf{valid}} \cup \mathbf{T}_{\mathbf{Parents}}$
14.             $\mathbf{T}_{\mathbf{valid}} = \mathbf{T}_{\mathbf{valid}} \cup \{\mathrm{T}\}$
15.             $\mathbf{T}_{\mathbf{valid}} = \mathbf{T}_{\mathbf{valid}} \cup SelectEventChildTables(\mathrm{T}, \mathbf{T}_{\mathbf{Ignore}}, \mathbf{T}_{\mathbf{Events}}, \mathfrak{S}_A)$
16.     Return $\mathbf{T}_{\mathbf{valid}}$

17. $\mathbf{T}_{\mathbf{Event}} = \{\mathrm{T}_{\mathrm{ET}}\} \cup SelectEventChildTables(\mathrm{T}_{\mathrm{ET}}, \mathbf{T}_{\mathbf{main}}, \emptyset, \mathfrak{S}_A)$
```

D.II Alternative main table selection

The most basic selection of event columns assumes that events should also be generated for parent tables of the main table. This may not be desirable, since this could introduce duplicate events because a single value may be used for multiple instances. Subsection Assign attributes of section 4.2.1 describes a basic approach to handle this, but this results in the incorrect exclusion of event columns in parent tables for which the values are not shared between artifact instances.

The algorithm below solves this by splitting $\mathbf{T}_{\mathbf{instance}}$ into a set of tables for which each record is associated with exactly one instance identifier $\mathbf{T}_{\mathbf{instanceEvents}}$ and a set of tables that contain records that are linked to multiple instance identifiers $\mathbf{T}_{\mathbf{instanceAttributes}}$. The idea is that events should only be generated based for records in $\mathbf{T}_{\mathbf{instanceEvents}}$; the records in $\mathbf{T}_{\mathbf{instanceAttributes}}$ should only be used to generate instance level attributes. The *Fraction(T, T_0)* function should return the *matched average fraction* between the two tables as defined in the related work subsection of section 3.3, while the *MatchedAvgFanout(T, T_0)* should return the *matched average fanout* as defined in the same subsection.

```
0.  $SelectMainInstanceParents(T_0, \mathfrak{S}_A)$
1.      $\mathbf{T}_{valid} = \emptyset$
2.      For each $T \in DirectParents(T_0, \mathfrak{S}_A)$
3.          If $Fraction(T, T_0) = 1 \wedge MatchedAvgFanout(T, T_0) = 1$ Then
4.              $\mathbf{T}_{valid} = \mathbf{T}_{valid} \cup \{T\}$
5.              $\mathbf{T}_{valid} = \mathbf{T}_{valid} \cup SelectMainInstanceParents(T, \mathfrak{S}_A)$
6.      Return $\mathbf{T}_{valid}$

7.  $SelectMainInstanceChildren(\mathbf{T}_0, \mathfrak{S}_A)$
8.      $\mathbf{T}_{valid} = \emptyset$
9.      For each $T_0 \in \mathbf{T}_0$
10.         For each $T \in DirectChildren(T_0, \mathfrak{S}_A)$
11.             If $\neg$(Event columns in T) $\wedge MatchedAvgFanout(T, T_0) = 1$ Then
12.                 $\mathbf{T}_{valid} = \mathbf{T}_{valid} \cup \{T\}$
13.                 $\mathbf{T}_{valid} = \mathbf{T}_{valid} \cup SelectMainInstanceChildren(\{T\}, \mathfrak{S}_A)$
14.     Return $\mathbf{T}_{valid}$

15. $\mathbf{T}_{instanceEvents} = SelectMainInstanceParents(T_m, \mathfrak{S}_A)$
16. $\mathbf{T}_{instanceEvents} = \mathbf{T}_{instanceEvents} \cup SelectMainInstanceChildren(\mathbf{T}_{instanceEvents}, \mathfrak{S}_A)$
17. $\mathbf{T}_{instanceAttributes} = SelectInstanceChildTables(T_m, \mathbf{T}_{instanceEvents}, \emptyset, \mathfrak{S}_A)$
18. $\mathbf{T}_{instanceAttributes} = \mathbf{T}_{instanceAttributes} \cup AllParents(\mathbf{T}_{instanceAttributes}, \mathfrak{S}_A)$
19. $\mathbf{T}_{instance} = \mathbf{T}_{instanceEvents} \cup \mathbf{T}_{instanceAttributes}$
```

As a result of the algorithm above $\mathbf{T}_{\mathbf{instanceEvents}}$ will contain all tables that could be merged into a single table with the same primary key as the given main table without introducing redundant values. The only exception are child tables that contain event columns for reasons explained below. Technically, if we take $\mathbf{C}_{\mathbf{pm}}$ to be the primary key of the given main artifact table, $\mathbf{C}_{\mathbf{X}}$ as the set of all columns in T_X and $\mathbf{C}_{\mathbf{px}}$ as the primary key of table T_X then:

$$\forall_{T_X \in \mathbf{T}_{instance}} \exists_{\mathbf{C}_{pm}} \left[[\mathbf{C}_{pm} \rightarrow \mathbf{C}_X] \wedge \forall_{\mathbf{C}_{px}} [\mathbf{C}_{pm} \rightarrow \mathbf{C}_{px}] \right] \qquad \mathbf{(6)}$$

The last part of the equation excludes (parent) tables that would introduce duplicate values when merged with the main artifact table.

If the *¬(Event columns in T)* statement on line 10 is removed all columns that are functionally determined by the primary key of the main table will be mapped to attributes of the trace, including columns in child tables that contain events. This may be undesirable since it can be argued that these columns should be mapped as attributes to the event columns in the child table.

If the *Fraction(T, T_0) = 1* statement on line 3 is removed then $\mathbf{T}_{\mathbf{instance}}$ will also contain parent tables that have a unique matching record for each record in the main table, but also contain records for which no matching record exists in the main table. For these additional table condition (6) does not hold, unless the non-matching records are removed. Since the extraction process only includes the matching records it may be desirable to keep these tables.

Appendix E: Quote schema-to-log mapping

```
<LogMapping logName="Quote log">
  <classifiers>
    <XesClassifier name="Unique classifiers" keys="concept:name event_id"/>
    <XesClassifier name="Activity classifiers" keys="concept:name"/>
  </classifiers>
  <extensions>
    <XesExtension name="Time" prefix="time" URI="http://www.xes-
               standard.org/time.xesext"/>
    <XesExtension name="Concept" prefix="concept" URI="http://www.xes-
               standard.org/concept.xesext"/>
  </extensions>
  <traceGlobalAttributes>
    <Attribute key=String datatype=String value=""/>
    <Attribute key=cdquote_order datatype=String value="MULTIPLE"/>
    <Attribute key=trace_id datatype=String value=""/>
    <Attribute key=Date datatype=Date value="01-01-1970"/>
    <Attribute key=cd_request datatype=String value="MULTIPLE"/>
    <Attribute key=inclusion datatype=String value="MULTIPLE"/>
    <Attribute key=Integer datatype=Integer value="0"/>
    <Attribute key=concept:name datatype=String value=""/>
  </traceGlobalAttributes>
  <eventGlobalAttributes>
    <Attribute key=time:timestamp datatype=Date value="01-01-1970"/>
    <Attribute key=event_id datatype=String value=""/>
    <Attribute key=Integer datatype=Integer value="0"/>
    <Attribute key=concept:name datatype=String value=""/>
  </eventGlobalAttributes>
  <traceMapping>
    <fromTable name="quote"/>
    <links>
      <Reference parentTable="request" childTable="quote">
        <columns>
          <ColumnPair parent="reqid" child="reqid"/>
        </columns>
      </Reference>
      <Reference parentTable="customer" childTable="request">
        <columns>
          <ColumnPair parent="name" child="customer_name"/>
        </columns>
      </Reference>
    </links>
    <generalAttributes>
      <AttributeMapping key=Date datatype=Date sourceColumn="request_date"/>
      <Attribute key=Integer datatype=Integer value="0">
        <AttributeMapping key=Price datatype=Integer sourceColumn="price"/>
        <AttributeMapping key=Reqid datatype=Integer sourceColumn="reqid"/>
      </Attribute>
      <AttributeMapping key=String datatype=String sourceColumn="name"/>
    </generalAttributes>
    <listAttributes>
      <ListAttribute key="cd_request">
        <fromTable reference="../../../fromTable"/>
        <links>
          <Reference reference="../../../../links/Reference"/>
          <Reference parentTable="request" childTable="cd_request">
            <columns>
              <ColumnPair parent="reqid" child="request_reqid"/>
            </columns>
          </Reference>
        </links>
        <generalAttributes>
          <AttributeMapping key=Integer datatype=Integer sourceColumn="quantity"/>
          <AttributeMapping key=String datatype=String sourceColumn="cd_name"/>
        </generalAttributes>
        <listAttributes/>
        <attributeId>
          <Column name="cd_name"/>
          <Column name="request_reqid"/>
        </attributeId>
      </ListAttribute>
      <ListAttribute key="cdquote_order">
        <fromTable reference="../../../fromTable"/>
        <links>
          <Reference parentTable="quote" childTable="cdquote_order">
            <columns>
              <ColumnPair parent="reqid" child="quote_reqid"/>
            </columns>
          </Reference>
        </links>
        <generalAttributes>
          <Attribute key=Integer datatype=Integer value="0">
            <AttributeMapping key=Deliverable_quantity datatype=Integer
                sourceColumn="deliverable_quantity"/>
            <AttributeMapping key=Order_orderid datatype=Integer
                sourceColumn="order_orderid"/>
            <AttributeMapping key=Quantity datatype=Integer
                sourceColumn="quantity"/>
          </Attribute>
          <AttributeMapping key=String datatype=String sourceColumn="cd_name"/>
        </generalAttributes>
        <listAttributes/>
        <attributeId>
          <Column name="cd_name"/>
          <Column name="quote_reqid"/>
          <Column name="order_orderid"/>
        </attributeId>
      </ListAttribute>
      <ListAttribute key="inclusion">
        <fromTable reference="../../../fromTable"/>
        <links>
          <Reference parentTable="quote" childTable="inclusion">
            <columns>
              <ColumnPair parent="reqid" child="quote_reqid"/>
            </columns>
          </Reference>
        </links>
        <generalAttributes>
          <AttributeMapping key=Integer datatype=Integer sourceColumn="quantity"/>
          <AttributeMapping key=String datatype=String sourceColumn="cd_name"/>
        </generalAttributes>
        <listAttributes/>
        <attributeId>
          <Column name="cd_name"/>
          <Column name="quote_reqid"/>
        </attributeId>
      </ListAttribute>
    </listAttributes>
    <traceId>
      <Column name="reqid"/>
    </traceId>
    <eventMappings>
      <EventMapping name="Reorder">
        <fromTable name="reorder"/>
        <links>
          <Reference parentTable="quote" childTable="reorder">
            <columns>
              <ColumnPair parent="reqid" child="quote_reqid"/>
            </columns>
          </Reference>
        </links>
        <generalAttributes>
          <Attribute key=Integer datatype=Integer value="0">
            <AttributeMapping key=Quote_reqid datatype=Integer
                sourceColumn="quote_reqid"/>
            <AttributeMapping key=Order_orderid datatype=Integer
                sourceColumn="order_orderid"/>
          </Attribute>
        </generalAttributes>
        <listAttributes/>
        <eventId>
          <Column name="quote_reqid"/>
          <Column name="order_orderid"/>
        </eventId>
        <eventColumn name="reorder_date"/>
      </EventMapping>
      <EventMapping name="Delivery customer accept shipment">
        <fromTable name="delivery"/>
        <links>
          <Reference parentTable="quote" childTable="delivery">
            <columns>
              <ColumnPair parent="reqid" child="quote_reqid"/>
            </columns>
          </Reference>
        </links>
        <generalAttributes>
          <AttributeMapping key=String datatype=String sourceColumn="delid"/>
        </generalAttributes>
        <listAttributes/>
        <eventId>
          <Column name="delid"/>
        </eventId>
        <eventColumn name="customer_accept_shipment_date"/>
      </EventMapping>
      <EventMapping name="Customer payment invoice issue">
        <fromTable name="customer_payment"/>
        <links>
          <Reference parentTable="quote" childTable="customer_payment">
            <columns>
              <ColumnPair parent="reqid" child="quote_reqid"/>
            </columns>
          </Reference>
        </links>
        <generalAttributes>
          <Attribute key=Integer datatype=Integer value="0">
            <AttributeMapping key=Price datatype=Integer sourceColumn="price"/>
            <AttributeMapping key=Quote_reqid datatype=Integer
                sourceColumn="quote_reqid"/>
          </Attribute>
        </generalAttributes>
        <listAttributes/>
        <eventId>
          <Column name="quote_reqid"/>
        </eventId>
        <eventColumn name="date_invoice_issue"/>
      </EventMapping>
      <EventMapping name="Customer payment sent">
        <fromTable reference="../../EventMapping[3]/fromTable"/>
        <links>
          <Reference reference="../../../EventMapping[3]/links/Reference"/>
        </links>
        <generalAttributes>
          <Attribute key=Integer datatype=Integer value="0">
            <AttributeMapping key=Price datatype=Integer sourceColumn="price"/>
            <AttributeMapping key=Quote_reqid datatype=Integer
                sourceColumn="quote_reqid"/>
          </Attribute>
        </generalAttributes>
        <listAttributes/>
        <eventId>
          <Column name="quote_reqid"/>
        </eventId>
        <eventColumn name="date_payment_sent"/>
      </EventMapping>
      <EventMapping name="Customer payment received">
        <fromTable reference="../../EventMapping[3]/fromTable"/>
        <links>
          <Reference reference="../../../EventMapping[3]/links/Reference"/>
        </links>
        <generalAttributes>
          <Attribute key=Integer datatype=Integer value="0">
            <AttributeMapping key=Price datatype=Integer sourceColumn="price"/>
            <AttributeMapping key=Quote_reqid datatype=Integer
                sourceColumn="quote_reqid"/>
          </Attribute>
        </generalAttributes>
        <listAttributes/>
        <eventId>
          <Column name="quote_reqid"/>
        </eventId>
        <eventColumn name="date_payment_received"/>
      </EventMapping>
      <EventMapping name="Opening">
        <fromTable name="quote"/>
        <links/>
        <generalAttributes>
          <AttributeMapping key=Integer datatype=Integer sourceColumn="reqid"/>
        </generalAttributes>
        <listAttributes/>
        <eventId>
          <Column name="reqid"/>
        </eventId>
        <eventColumn name="opening_date"/>
      </EventMapping>
```

```
<EventMapping name="Customer no deliverable notification">
  <fromTable reference="../../EventMapping[6]/fromTable"/>
  <links/>
  <generalAttributes>
    <AttributeMapping key=Integer datatype=Integer sourceColumn="reqid"/>
  </generalAttributes>
  <listAttributes/>
  <eventId>
    <Column name="reqid"/>
  </eventId>
  <eventColumn name="customer_no_deliverable_notification_date"/>
</EventMapping>
<EventMapping name="Rejection">
  <fromTable reference="../../EventMapping[6]/fromTable"/>
  <links/>
  <generalAttributes>
    <AttributeMapping key=Integer datatype=Integer sourceColumn="reqid"/>
  </generalAttributes>
  <listAttributes/>
  <eventId>
    <Column name="reqid"/>
  </eventId>
  <eventColumn name="rejection_quote_date"/>
</EventMapping>
<EventMapping name="Acceptance">
  <fromTable reference="../../EventMapping[6]/fromTable"/>
  <links/>
  <generalAttributes>
    <AttributeMapping key=Integer datatype=Integer sourceColumn="reqid"/>
  </generalAttributes>
  <listAttributes/>
  <eventId>
    <Column name="reqid"/>
  </eventId>
  <eventColumn name="acceptance_quote_date"/>
</EventMapping>
<EventMapping name="Order adding">
  <fromTable name="quote_order"/>
  <links>
    <Reference parentTable="quote" childTable="quote_order">
      <columns>
        <ColumnPair parent="reqid" child="quote_reqid"/>
      </columns>
    </Reference>
  </links>
  <generalAttributes>
    <Attribute key=Integer datatype=Integer value="0">
      <AttributeMapping key=Quote_reqid datatype=Integer
          sourceColumn="quote_reqid"/>
      <AttributeMapping key=Order_orderid datatype=Integer
          sourceColumn="order_orderid"/>
    </Attribute>
  </generalAttributes>
  <listAttributes/>
  <eventId>
    <Column name="quote_reqid"/>
    <Column name="order_orderid"/>
  </eventId>
  <eventColumn name="adding_date"/>
```

```
      </EventMapping>
    </eventMappings>
  </traceMapping>
</LogMapping>
```

Appendix F: Prototype implementation

A prototype was implemented in Java to demonstrate the approach. The prototype is available at http://dl.dropbox.com/u/18902457/Prototype.zip. This zip file contains the example files of the CD shop in the "CD Shop files" folder, the sources including the Eclipse project in the "Sources" folder and a compiled version of the prototype (the "Data2Events.jar" file and the "Data2Events_lib" folder).

F.I Overview

Architecture

The prototype is implemented as a layered application based on the Data Access Object Pattern[17]. Figure 29 shows an overview of the relation between the main packages, except for the util package.

- The *util* package does not depend on any other code (in the project package). This package contains general helper methods that could be useful in any application.
- The *model* package depends only on the util package. The classes in this package are (data) transfer objects[18] that do not contain any logic.
- The *data* package depends only on the util and model packages. This package contains the actual data access objects that wrap the storage implementation.
- The *algorithms* package depends only on the util, model and data packages. The classes in this package contain the logic of the prototype.
- The *UI* package depends on all other sub-packages (but not on any code in the main package). The classes in this package only expose the logic in the algorithms package to the user.

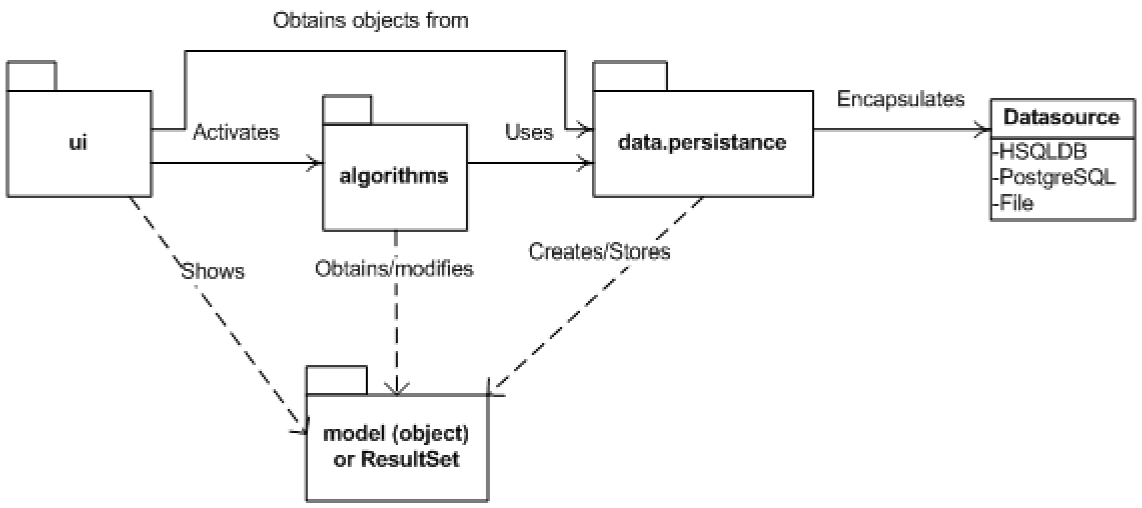


Figure 29: Prototype architecture

Some sub-packages also contain a Util class. In this case other classes in the same sub-package may use the Util class, but the Util class will not depend on any other code in the same sub-package.

Starting the application and database connections

In windows the application can be started using the "start Data2Events.bat" file; for other platforms this file also shows the required parameters. Note that the maximum

[17] http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html
[18] http://martinfowler.com/eaaCatalog/dataTransferObject.html

memory size was set to 500 MB to improve performance in general, but a significantly lower value is likely sufficient for the CD shop example.

When starting the application a dialog asks for the connection parameters for an embedded HSQL database. This will connect to an existing database if one exists at the given path or create a new database if no database exists at the location (no installation required). The default path is the "internal db" subfolder (with "hsql.*" as the database file names).

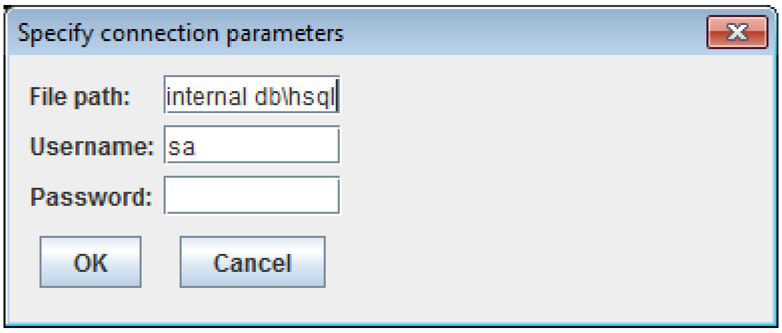


Figure 30: HSQLDB connection parameters

A different internal database can be selected from the *Settings > Storage mechanism* menu as shown below. There are 3 options:

- **File based**: This is meant for the transfer of data and metadata only. When this is used a number of xml and csv files will be created at the given location. These files can then easily be copied to another system and imported again as described in section F.II below.
- **HSQLDB**: A fast embedded database with the capability to store up to 16 GB of data. The prototype will create the database (if it does not exist yet) and the required structure (e.g. tables).
- **PostgreSQL**: A database suitable for larger datasets. To use this a PostgreSQL 9.0 database needs to be installed and the database used by the prototype must exist in PostgreSQL. The prototype will create the required structure in the database though.

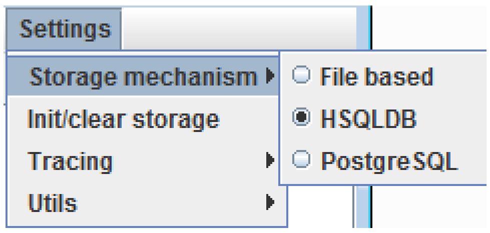
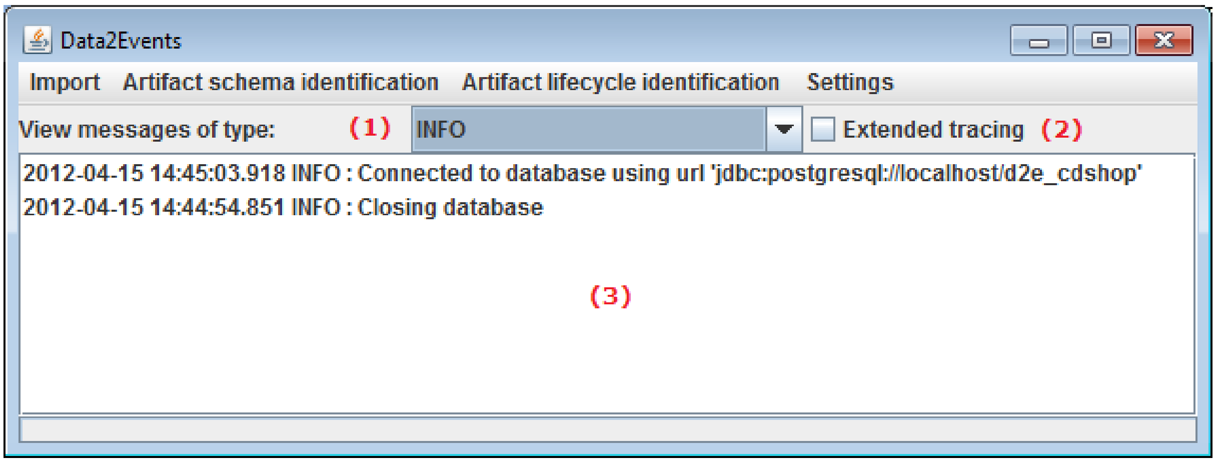


Figure 31: Internal storage mechanisms

<u>Before any storage mechanism can be used it needs to be initialized</u> using the *Settings > Init/clear storage* menu option. This will create the structure required for the storage mechanism to function. Note that this will remove any existing data in the storage.

Trace information

During the execution of the application various trace information will be written to the screen and log files. In the application this is shown in the area marked by (3) in Figure 32. The settings in the main screen define (1) what kind of messages are shown and (2) how detailed the information on the screen should be. Note that the type of messages shown on the screen cannot be more detailed than the messages written to the log files (as described below).



```
2012-04-15 14:45:03.918 INFO : Connected to database using url 'jdbc:postgresql://localhost/d2e_cdshop'
2012-04-15 14:44:54.851 INFO : Closing database
```

Figure 32: Prototype trace information

All trace messages are written to log files in the "\log" subfolder of the application. At most 10 files of 5 MB each will be created per day. The types of messages written to log files is defined by the *Settings* > *Tracing* > *Source trace level* menu item. The parameters of method calls can also be written to the log files. By default this is only done for error messages, but this can be changed to include other types of messages with the *Settings* > *Tracing* > *Parameter trace level* menu item.

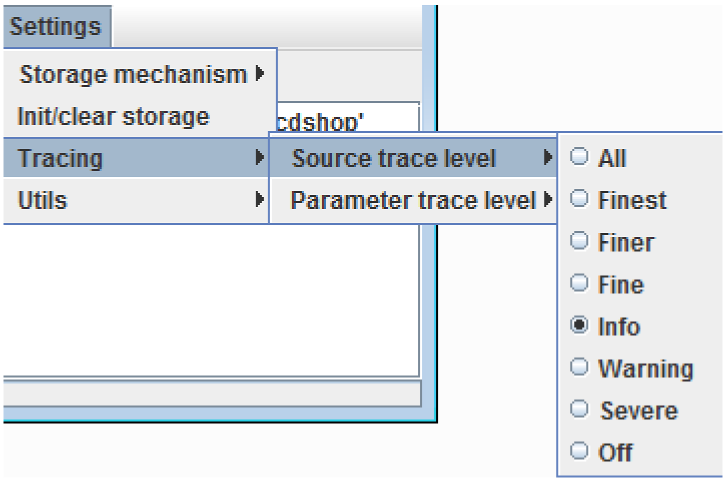


Figure 33: Prototype trace settings

F.II Data transfer

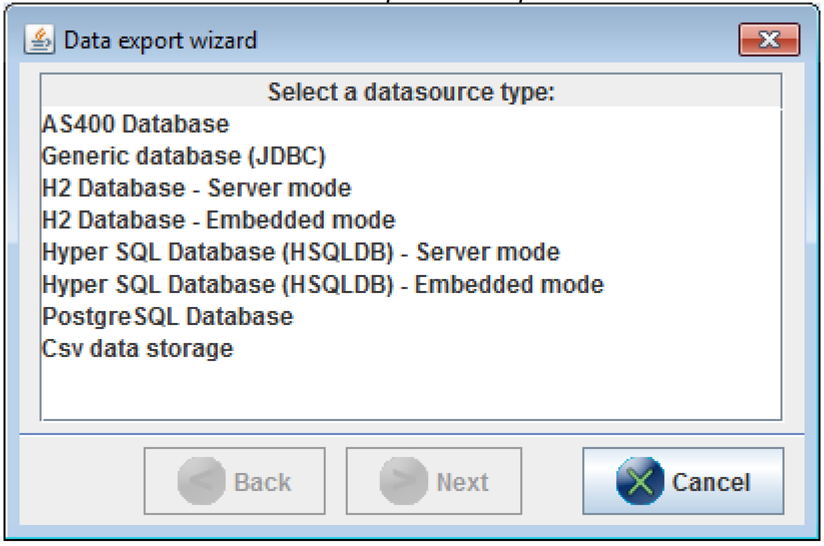
The starting point for the approach is a dataset with any metadata that is already known. The prototype was set up to import this information once so the source system would not have to be available for the remainder of the approach. This is done in a number of steps:

1. Setting the storage mechanism to *file based* on the source system and importing the data and metadata as described below.
2. Copying the created files to the system where the dataset will be analyzed.
3. Settings the storage mechanism to *HSQLDB* or *PostgreSQL* and importing the data and metadata from the files as described below.

Note that it is be possible to import the data and metadata from the source system directly to a *HSQLDB* or *PostgreSQL* storage, but in that case the information would obviously end up on the same system. More detailed information on data transfer is available in the javadoc documentation of the *ui.transfer*, *data* and *data.dataImport* packages.

For the CD shop example the first two steps were already done. The files are available in the "CD Shop files" folder. They can be imported with the following steps:

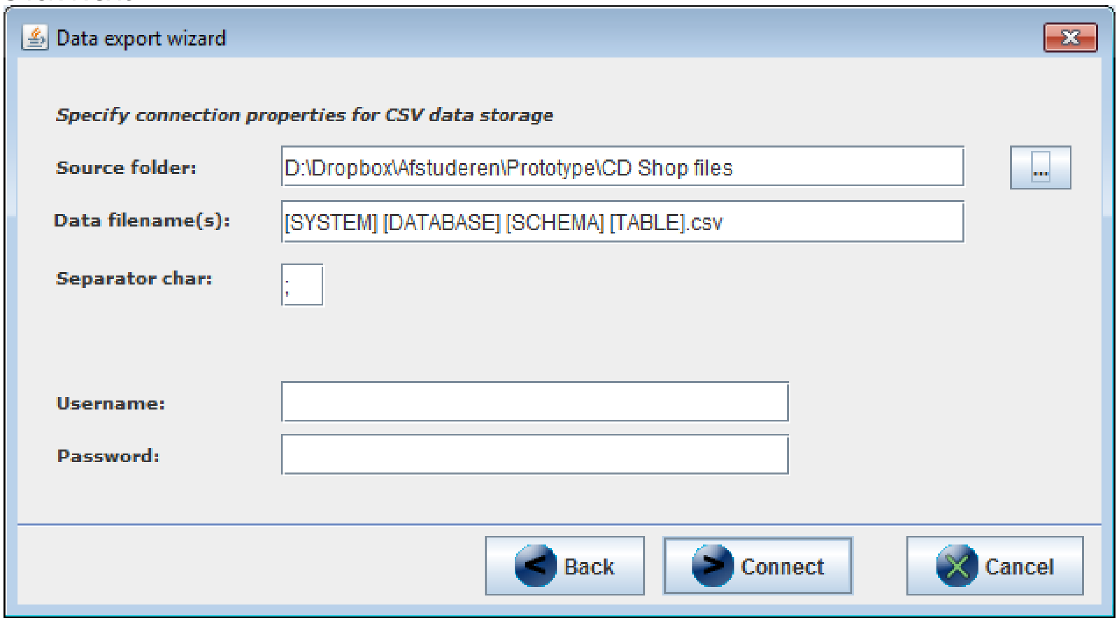
1. Start the wizard from the *Import > Import data & metadata* menu item.



Select a datasource type:

AS400 Database
Generic database (JDBC)
H2 Database - Server mode
H2 Database - Embedded mode
Hyper SQL Database (HSQLDB) - Server mode
Hyper SQL Database (HSQLDB) - Embedded mode
PostgreSQL Database
Csv data storage

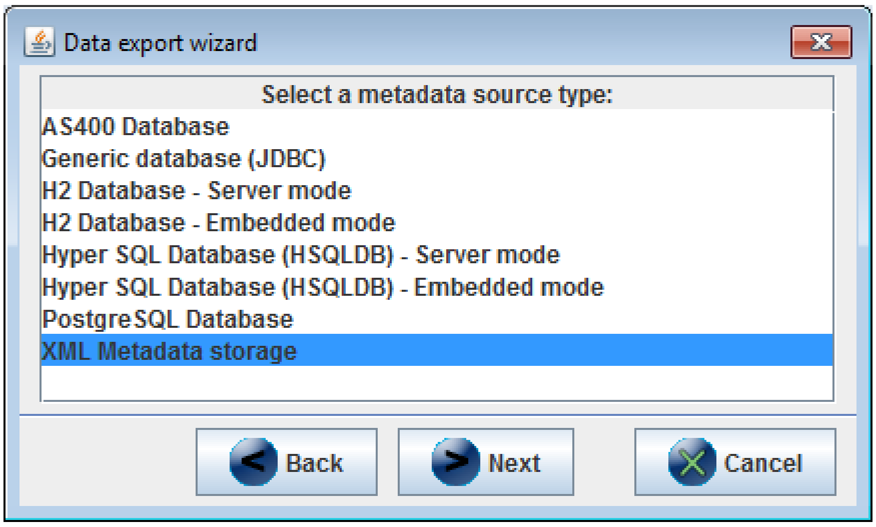
2. Select "Csv data storage".
3. Click *Next*.



Specify connection properties for CSV data storage

Source folder:	D:\Dropbox\Afstuderen\Prototype\CD Shop files
Data filename(s):	[SYSTEM] [DATABASE] [SCHEMA] [TABLE].csv
Separator char:	;
Username:	
Password:	

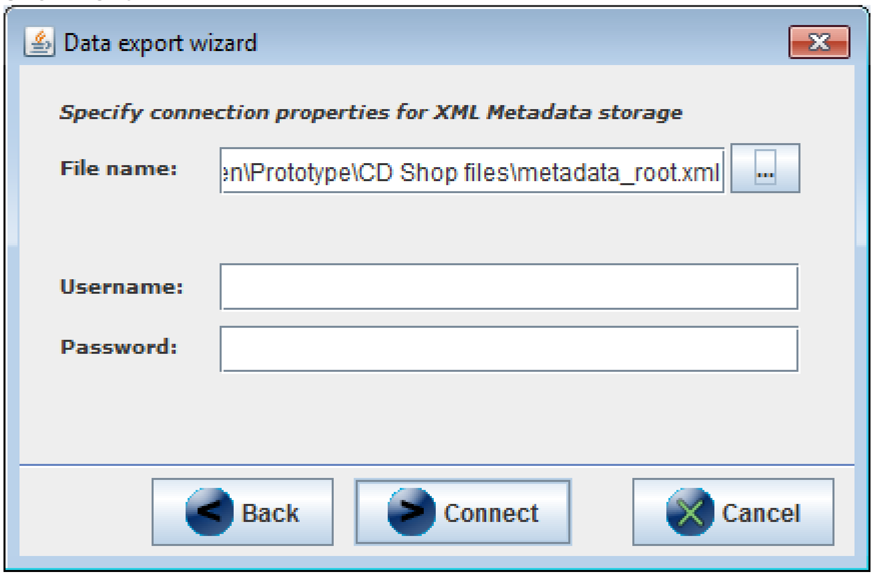
4. Enter the path to the "CD shop files" folder (possibly via the "…" button) as the *source folder*.
5. Click *Connect*.


Data export wizard

Select a metadata source type:

AS400 Database
Generic database (JDBC)
H2 Database - Server mode
H2 Database - Embedded mode
Hyper SQL Database (HSQLDB) - Server mode
Hyper SQL Database (HSQLDB) - Embedded mode
PostgreSQL Database
XML Metadata storage

6. Select "XML Metadata storage".
7. Click *Next*.


Data export wizard

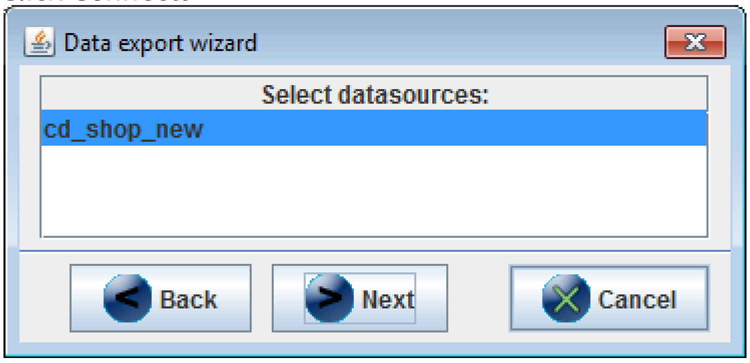
Specify connection properties for XML Metadata storage

File name: en\Prototype\CD Shop files\metadata_root.xml

Username:

Password:

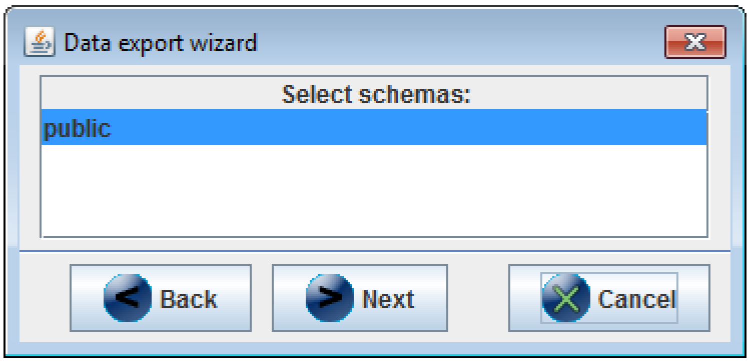
8. Enter the path to the "metadata_root.xml" file in the "CD shop files" folder (possibly via the "…" button) as the *file name*.
9. Click *Connect*.


Data export wizard

Select datasources:

cd_shop_new

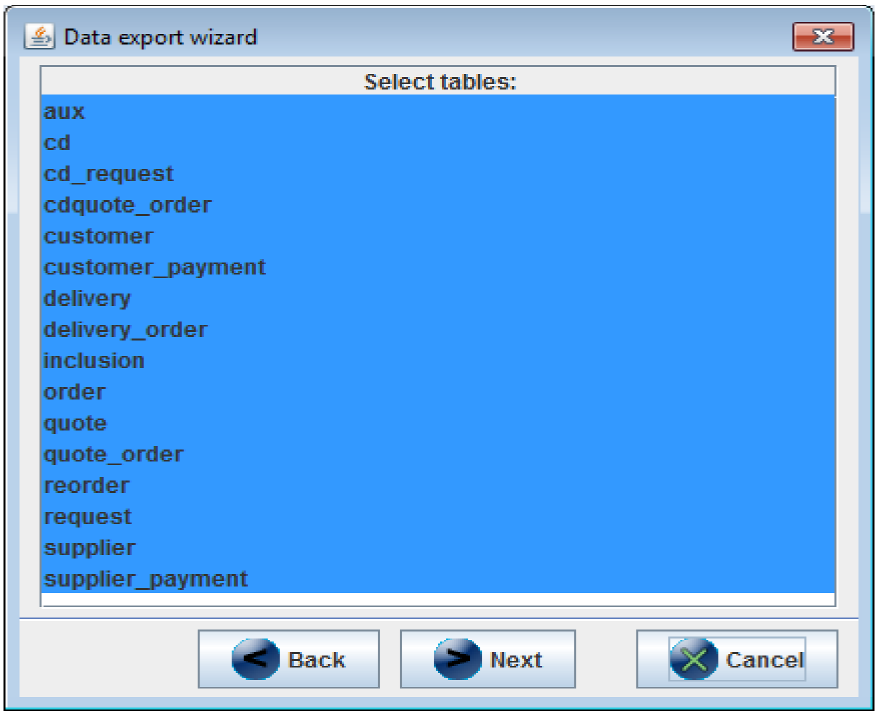
10. Click *Next*.


Data export wizard

Select schemas:

public

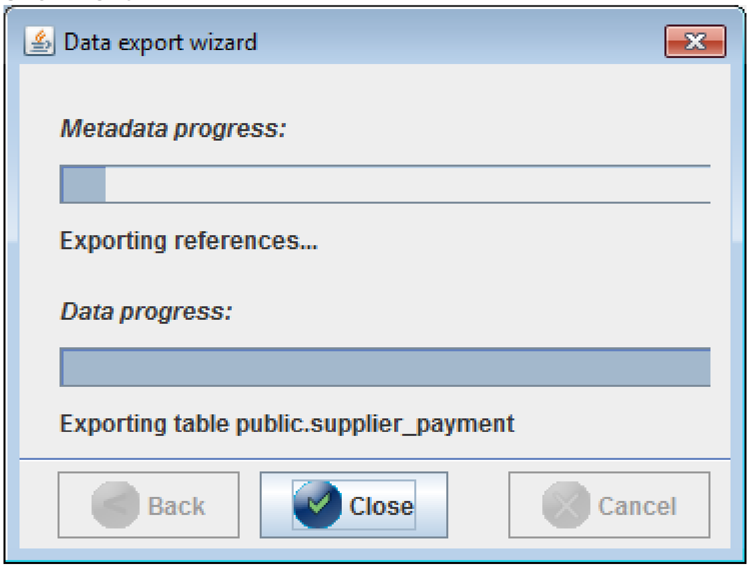
11. Click *Next*.


Data export wizard

Select tables:

aux
cd
cd_request
cdquote_order
customer
customer_payment
delivery
delivery_order
inclusion
order
quote
quote_order
reorder
request
supplier
supplier_payment

12. Click *Next*.


Data export wizard

Metadata progress:

Exporting references...

Data progress:

Exporting table public.supplier_payment

13. Wait until the import is complete and click *Close*.

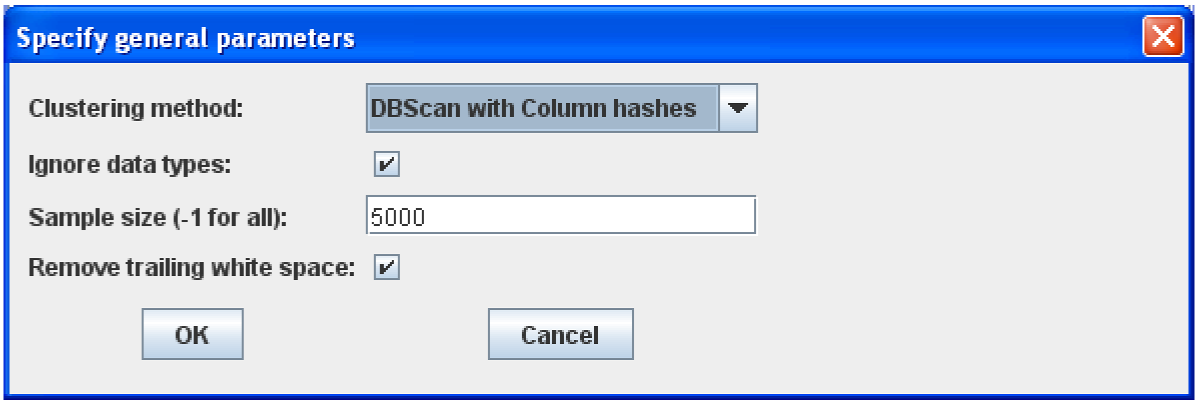
F.III Artifact schema identification

F.III.I Schema extraction

All techniques used in the schema extraction step can be found in the *Artifact schema identification > Schema extraction* menu. These will only work if the storage mechanism is initialized and data is imported as described in the previous sections.

Domain extraction

Domain extraction can be activated by the *Determine column domains* menu item. A series of dialogs is shown to gather the required parameters.



Specify general parameters

Clustering method: DBScan with Column hashes

Ignore data types: ✓

Sample size (-1 for all): 5000

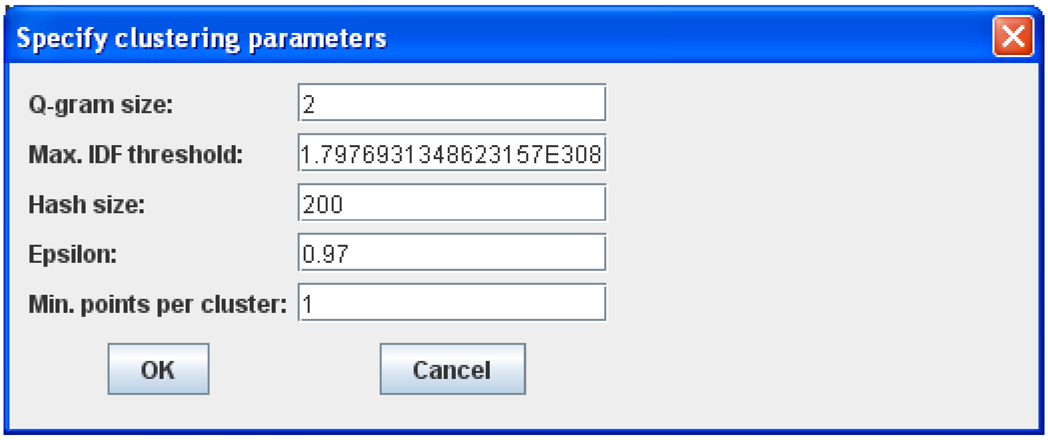
Remove trailing white space: ✓

Figure 34: General domain extraction parameters dialog

The first dialog requests general domain extraction parameters:

- The *Clustering method* to be used:
    - "Datatype only" specificies that the XES datatypes will be determined heuristically, but no further domain clustering will be done. This setting was used as the basis for the results in the empirical evaluation in the steps following domain extraction.
    - "DBScan with column hashes" specifies that column distances will be calculated through q-gram min-hashes while the actual clustering will be done with the DBScan algorithm.
    - "DBScan with PCA-T" specifies that column distances will be calculated through column signatures obtained by principal component analysis (PCA) on the column x column covariance matrix while the actual clustering will be done with the DBScan algorithm.
    - "K-Center with column hashes" specifies that column distances will be calculated through q-gram min-hashes while the actual clustering will be done with the k-center algorithm.
    - "K-Center with PCA-T" specifies that column distances will be calculated through column signatures obtained by principal component analysis (PCA) on the column x column covariance matrix while the actual clustering will be done with the k-center algorithm.
- *Ignore datatypes* specifies if the JDBC datatypes can be used to heuristically determine the XES datatypes. For the CD shop example the JDBC datatypes are not trustworthy and should therefore be ignored.
- *Sample size* specifies the maximum number of records that should be used for the domain extraction process. This value is used for both the heuristic XES datatype determinination and the actual clustering approach.
- *Remove trailing white space* can be used to properly handle fixed-length character values stored in databases. If trailing white space would not be removed in these

cases then resemblance between values would likely be much higher than desirable since each value likely contains a significant amount of whitespace.


Specify clustering parameters

Q-gram size: 2

Max. IDF threshold: 1.7976931348623157E308

Hash size: 200

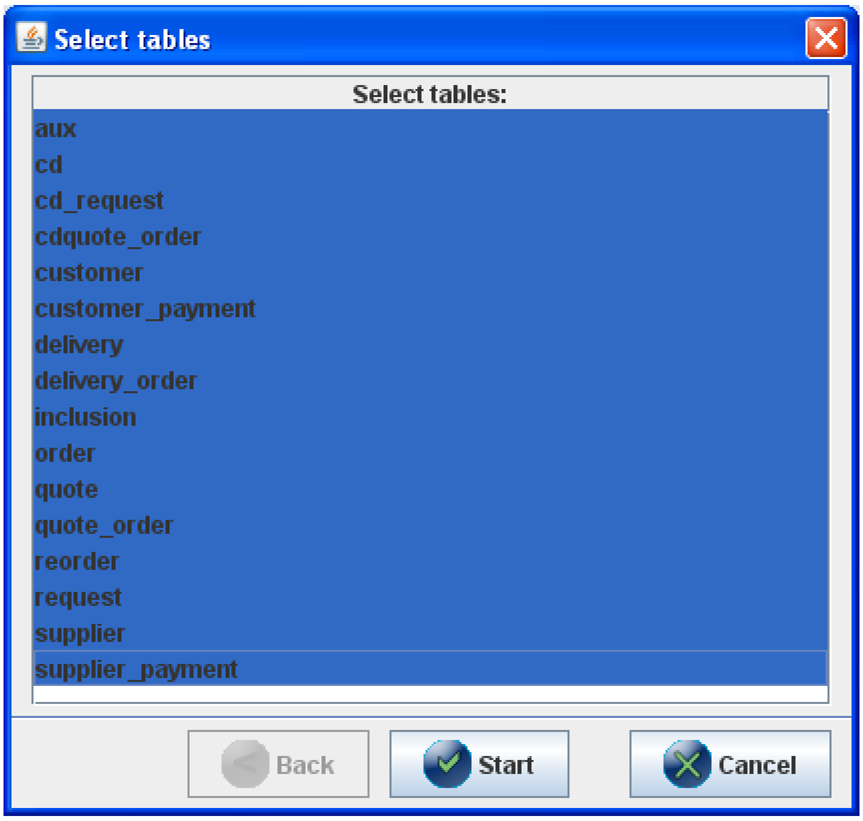
Epsilon: 0.97

Min. points per cluster: 1

Figure 35: Example domain clustering method parameters dialog

If any *clustering method* other than "Datatypes only" is chosen the second dialog will request the parameters for the clustering method:

- *Q-gram size* specifies the number of characters in each q-gram extracted.
- *Maximum IDF threshold* specifies the maximum inverse document frequency a q-gram can have to be included in the distance calculation. The default value includes all q-grams.
- *Hash size* specifies the number of values that are used to calculate min-hash distances between columns. This is only relevant (and displayed) if a clustering method is chosen that uses these.
- *Variance fraction* specificies the fraction of the total variance to use when selecting the number of principal components to calculate the column signatures with the the column x column covariance matrix. This is only relevant (and displayed) if a clustering method is used that calculates distances through principal component analysis.
- *Epsilon* specifies the maximum range when adding new column nodes to a domain cluster as defined by the DBScan algorithm. This is only relevant (and displayed) if a clustering method based on DBScan is chosen.
- *Minumum points per cluster* specifies the minimum number of other column nodes that need to be within range of a column to form a new domain cluster. Effectively this means that all domain clusters will contain at least *minimum points per cluster* + 1 column nodes or exactly one column (in case of isolated columns that are classified as noise by the DBScan algorithm). This is only relevant (and displayed) if a clustering method based on DBScan is chosen.
- *Number of clusters per datatype* specifies the number of domain clusters generated for each XES datatype by the k-center algorithm. This is only relevant (and displayed) if a clustering method based on k-center is chosen.



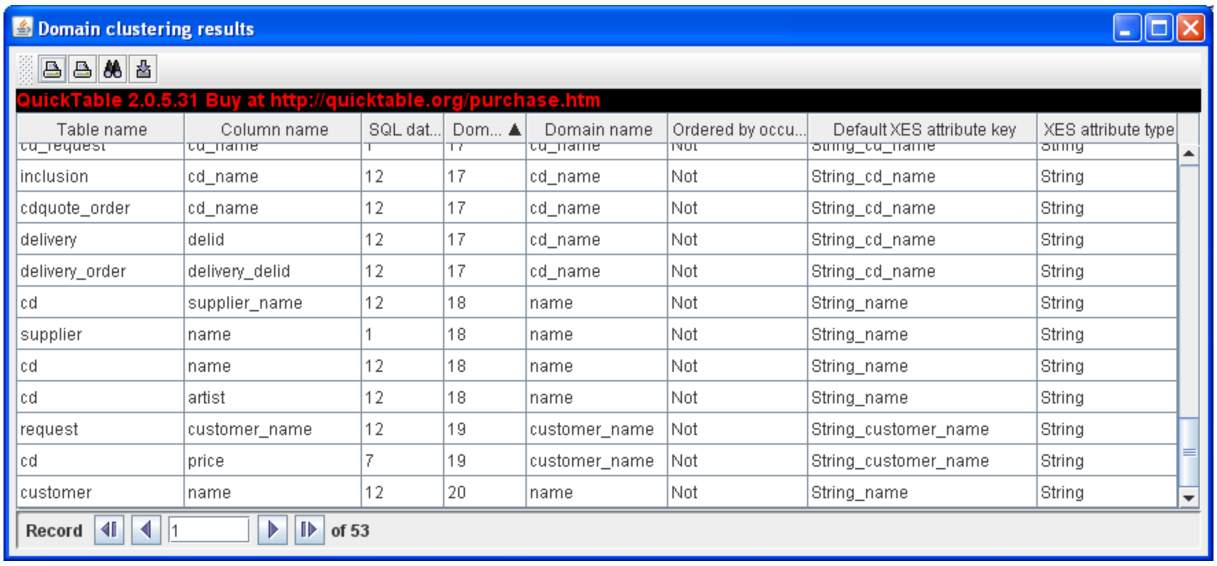
Select tables:

aux
cd
cd_request
cdquote_order
customer
customer_payment
delivery
delivery_order
inclusion
order
quote
quote_order
reorder
request
supplier
supplier_payment

Figure 36: Select tables dialog

The final dialog will request for which tables the column domains should be extracted. After this dialog the domain extraction process will start. During the extraction process temporary files will be created in the default temporary file location of the system or user.

The results of the domain extraction can be viewed using the *View domain clustering results* menu item. This opens the screen shown below (after a commercial dialog).



QuickTable 2.0.5.31 Buy at http://quicktable.org/purchase.htm

Table name	Column name	SQL dat...	Dom... ▲	Domain name	Ordered by occu...	Default XES attribute key	XES attribute type
cd_request	cd_name	1	17	cd_name	Not	String_cd_name	String
inclusion	cd_name	12	17	cd_name	Not	String_cd_name	String
cdquote_order	cd_name	12	17	cd_name	Not	String_cd_name	String
delivery	delid	12	17	cd_name	Not	String_cd_name	String
delivery_order	delivery_delid	12	17	cd_name	Not	String_cd_name	String
cd	supplier_name	12	18	name	Not	String_name	String
supplier	name	1	18	name	Not	String_name	String
cd	name	12	18	name	Not	String_name	String
cd	artist	12	18	name	Not	String_name	String
request	customer_name	12	19	customer_name	Not	String_customer_name	String
cd	price	7	19	customer_name	Not	String_customer_name	String
customer	name	12	20	name	Not	String_name	String

Figure 37: Domain clustering results screen

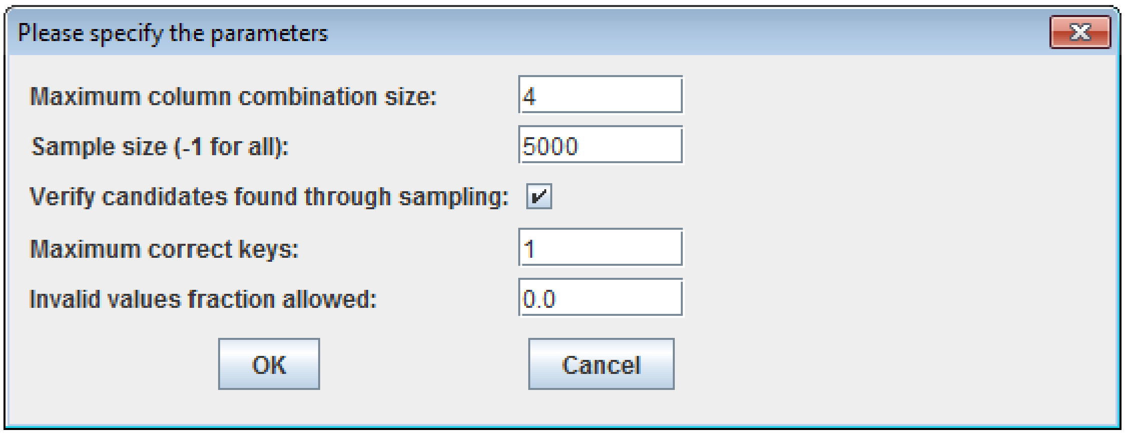
Most columns in the domain clustering result screen are self-explanatory. The SQL datatype column shows the JDBC type number. Common type numbers are: char(1), varchar (12), numeric (2), decimal (3), float (6), real (7), double (8), integer (4), boolean (16), date (91), time (92) and timestamp (93).

-

More detailed information about the implementation is available in the javadoc documentation of the *algorithms.domaincategorization*, *algorithms.clustering*, *model.clustering* and *algorithms.qGramExtraction* packages.

Primary key extraction

Primary key extraction can be activated by the *Determine primary keys* menu item. A dialog is shown to gather the specific primary key extraction parameters followed by the table selection dialog shown in Figure 36.


Please specify the parameters

Maximum column combination size: 4

Sample size (-1 for all): 5000

Verify candidates found through sampling: ✓

Maximum correct keys: 1

Invalid values fraction allowed: 0.0

Figure 38: Primary key extraction parameters dialog

The dialog requests the following parameters:

- *Maximum column combination size* specificies the number of columns a candidate key can contain. Column combinations that consist of more columns are not tested (and thus not returned) by the HCA algorithm.
- *Sample size* specifies the maximum number of records to use when determining candidate keys. Note that the Gordian part of the candidate key extraction algorithm was implemented to never use more than 1 000 samples, but will use a smaller number if specified by the sample size.
- *Verify candidates found through sampling* specificies if the candidate keys found by the HCA-Gordian algorithm should be verified using the complete dataset. Note that this parameter will have no effect for tables for which the sample size is larger the number of records they contain.
- *Maximum correct keys* specifies the number of primary key candidates to extract. Less primary key candidates may be extracted if there are insufficient candidate keys found by the HCA-Gordian algorithm.
- *Invalid values fraction allowed* specifies the fraction of records that are allowed to violate the candidate key when it is verified. Note that this parameter is only used by the verification step, thus the HCA-Gordian algorithm may still remove possible candidate keys even if the number of violating records for that key is less than the given fraction of the total number of records.

The results of the primary key extraction can be viewed using the *View primary keys results* menu item. This opens the screen shown below (after a commercial dialog).

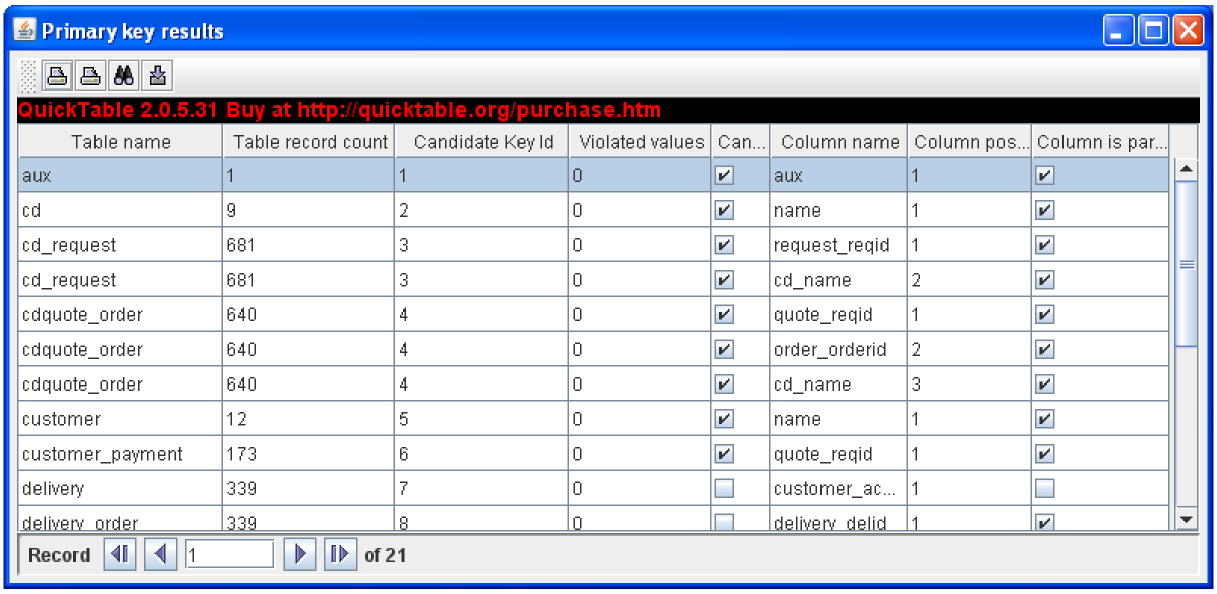

Primary key results
QuickTable 2.0.5.31 Buy at http://quicktable.org/purchase.htm

Table name	Table record count	Candidate Key Id	Violated values	Can...	Column name	Column pos...	Column is par...
aux	1	1	0	✓	aux	1	✓
cd	9	2	0	✓	name	1	✓
cd_request	681	3	0	✓	request_reqid	1	✓
cd_request	681	3	0	✓	cd_name	2	✓
cdquote_order	640	4	0	✓	quote_reqid	1	✓
cdquote_order	640	4	0	✓	order_orderid	2	✓
cdquote_order	640	4	0	✓	cd_name	3	✓
customer	12	5	0	✓	name	1	✓
customer_payment	173	6	0	✓	quote_reqid	1	✓
delivery	339	7	0	☐	customer_ac...	1	☐
delivery_order	339	8	0	☐	delivery_delid	1	✓

Record 1 of 21

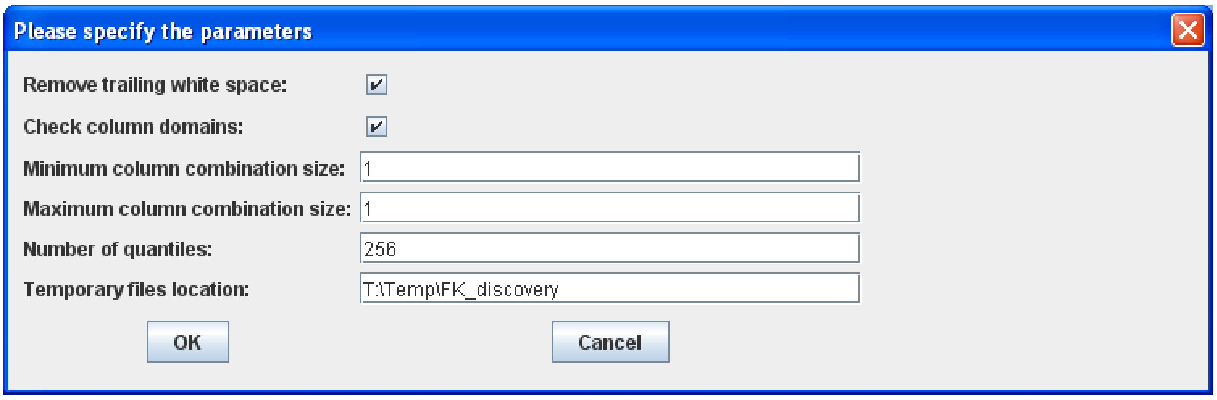
Figure 39: Primary key extraction results screen

Most columns in the primary key extraction result screen are self-explanatory. *Violated values* shows the number of records that violated the candidate key. *Candidate is Primary Key* shows if the candidate key is defined as the primary key on the table. Finally *Column is part of key* shows if the candidate key column is part of the primary key of the table.

More detailed information about the implementation is available in the javadoc documentation of the *algorithms.pkdiscovery* package and subpackages.

Foreign key extraction

Foreign key extraction can be activated by the *Determine foreign keys* menu item. A dialog is shown to gather the specific foreign key extraction parameters followed by the table selection dialog shown in Figure 36.



Remove trailing white space:	✓
Check column domains:	✓
Minimum column combination size:	1
Maximum column combination size:	1
Number of quantiles:	256
Temporary files location:	T:\Temp\FK_discovery

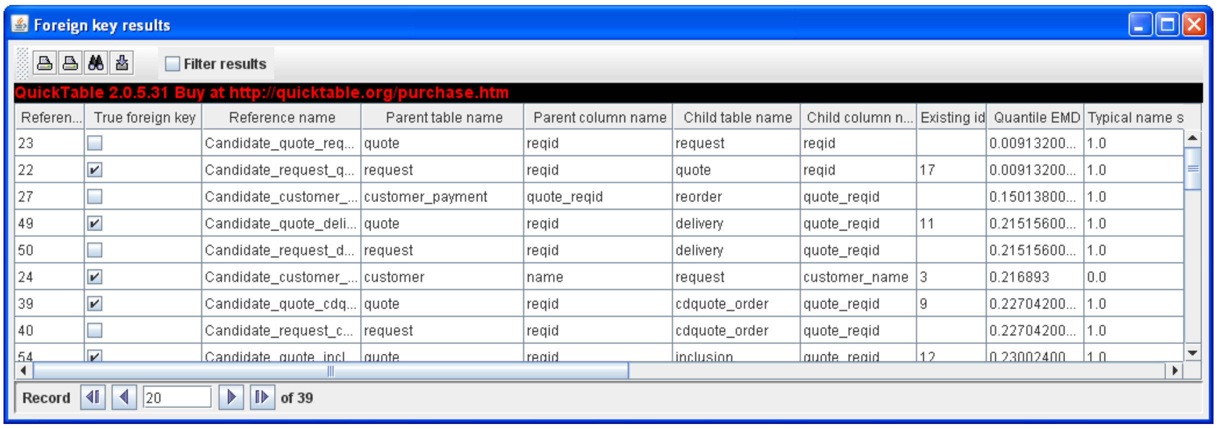
Figure 40: Foreign key extraction parameters dialog

The dialog requests the following parameters:

- *Remove trailing white space* can be used to properly handle fixed-length character values stored in databases. If this is checked then values that differ only in the amount of trailing whitespace will be treated as identical.
- *Check column domains* specificies is candidate inclusion dependencies (IND) should be pruned based on the domains of the columns or column combinations.
- *Minimum column combination size* and *maximum column combination size* specify the number of columns that can be on the left-hand and right-hand side of each IND to verify.

- *Number of quantiles* specifies the total number of “buckets” to use for the comparison of the value distribution histogram of the child and parent column combination of an IND.
- *Temporary files location* specifies where temporary files will be created while the algorithm is running. The default location is the standard temporary file location of the system or user.

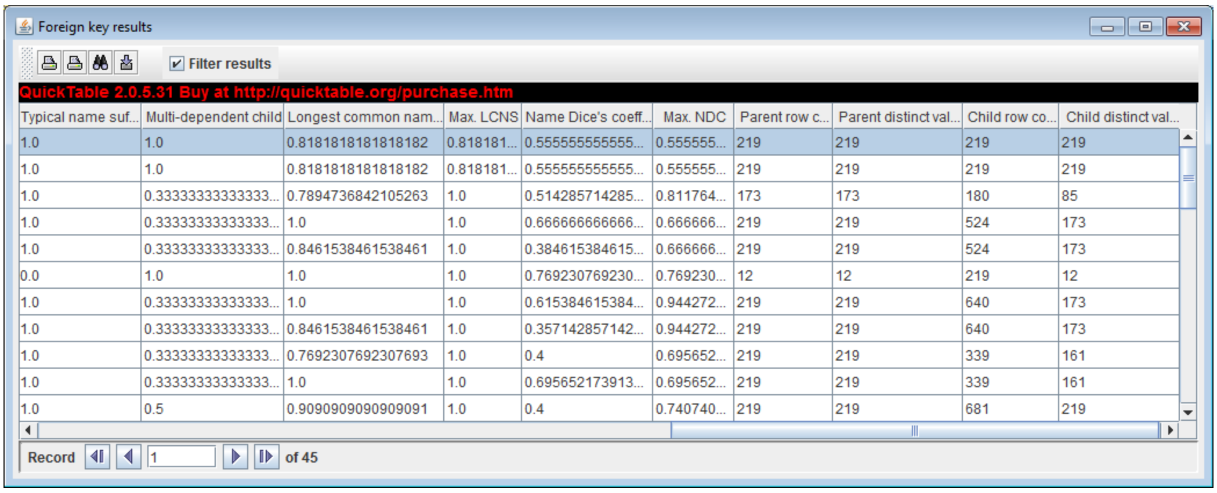
The results of the foreign key extraction can be viewed using the *View foreign keys results* menu item. This opens the screen shown below (after a commercial dialog). Note that the pictures below are both parts of the same screen. The *Filter results* checkbox can be used to hide all candidate foreign keys for which the *Quantile EMD* value is greater than 1 or for which the LCNS score is not the highest LCNS score for that child column combination.


Foreign key results
Filter results
QuickTable 2.0.5.31 Buy at http://quicktable.org/purchase.htm

Referen...	True foreign key	Reference name	Parent table name	Parent column name	Child table name	Child column n...	Existing id	Quantile EMD	Typical name s
23	☐	Candidate_quote_req...	quote	reqid	request	reqid		0.00913200...	1.0
22	☑	Candidate_request_q...	request	reqid	quote	reqid	17	0.00913200...	1.0
27	☐	Candidate_customer_...	customer_payment	quote_reqid	reorder	quote_reqid		0.15013800...	1.0
49	☑	Candidate_quote_deli...	quote	reqid	delivery	quote_reqid	11	0.21515600...	1.0
50	☐	Candidate_request_d...	request	reqid	delivery	quote_reqid		0.21515600...	1.0
24	☑	Candidate_customer_...	customer	name	request	customer_name	3	0.216893	0.0
39	☑	Candidate_quote_cdq...	quote	reqid	cdquote_order	quote_reqid	9	0.22704200...	1.0
40	☐	Candidate_request_c...	request	reqid	cdquote_order	quote_reqid		0.22704200...	1.0
54	☑	Candidate_quote_incl	quote	reqid	inclusion	quote_reqid	12	0.23002400	1.0

Record 20 of 39

Figure 41: Foreign keys extraction results screen (left)


Foreign key results
Filter results
QuickTable 2.0.5.31 Buy at http://quicktable.org/purchase.htm

Typical name suf...	Multi-dependent child	Longest common nam...	Max. LCNS	Name Dice's coeff...	Max. NDC	Parent row c...	Parent distinct val...	Child row co...	Child distinct val...
1.0	1.0	0.8181818181818182	0.818181...	0.555555555555...	0.555555...	219	219	219	219
1.0	1.0	0.8181818181818182	0.818181...	0.555555555555...	0.555555...	219	219	219	219
1.0	0.33333333333333...	0.7894736842105263	1.0	0.514285714285...	0.811764...	173	173	180	85
1.0	0.33333333333333...	1.0	1.0	0.666666666666...	0.666666...	219	219	524	173
1.0	0.33333333333333...	0.8461538461538461	1.0	0.384615384615...	0.666666...	219	219	524	173
0.0	1.0	1.0	1.0	0.769230769230...	0.769230...	12	12	219	12
1.0	0.33333333333333...	1.0	1.0	0.615384615384...	0.944272...	219	219	640	173
1.0	0.33333333333333...	0.8461538461538461	1.0	0.357142857142...	0.944272...	219	219	640	173
1.0	0.33333333333333...	0.7692307692307693	1.0	0.4	0.695652...	219	219	339	161
1.0	0.33333333333333...	1.0	1.0	0.695652173913...	0.695652...	219	219	339	161
1.0	0.5	0.9090909090909091	1.0	0.4	0.740740...	219	219	681	219

Record 1 of 45

Figure 42: Foreign keys extraction results screen (right)

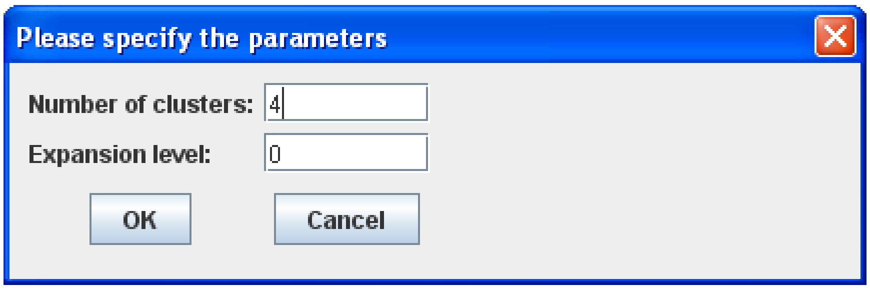
Most columns in the foreign key extraction result screen are self-explanatory, given the definitions of the evaluation properties in subsection 3.2.3. *True foreign key* shows if the candidate foreign key is defined as an actual foreign key in the schema. Note that this column can be updated so the foreign key is taken into account for further steps. *Existing id* shows the identifier of the corresponding actual foreign key. *Quantile EMD* shows the (thresholded) Earth Movers Distance (EMD) between the value distribution of the parent and child colums. *Maximum LCNS* shows the maximum LCNS score of the child column combination and all of its identified parent column combinations. Similarly *Maximum NDC* shows the maximum name Dice’s coefficient score of the child column combination and all of its identified parent column combinations.

More detailed information about the implementation is available in the javadoc documentation of the *algorithms.fkdiscovery*, *algorithms.fkdiscovery.quantiles* and *algorithms.emd* packages.

F.III.II Identify artifact schemas

All menu items related to this subject can be found in the *Artifact schema identification > Identify artifact schemas* menu. To identify artifact schemas 4 menu items must be activated in sequence:

1. *Calculate entropies*. No further input is required.
2. *Calculate table importances*. No further input is required.
3. *Calculate distances*. No further input is required.
4. *Cluster tables*. A dialog is shown to gather the specific table clustering parameters followed by the table selection dialog shown in Figure 36. Note that previously identified artifacts will be deleted when this menu item is activated.



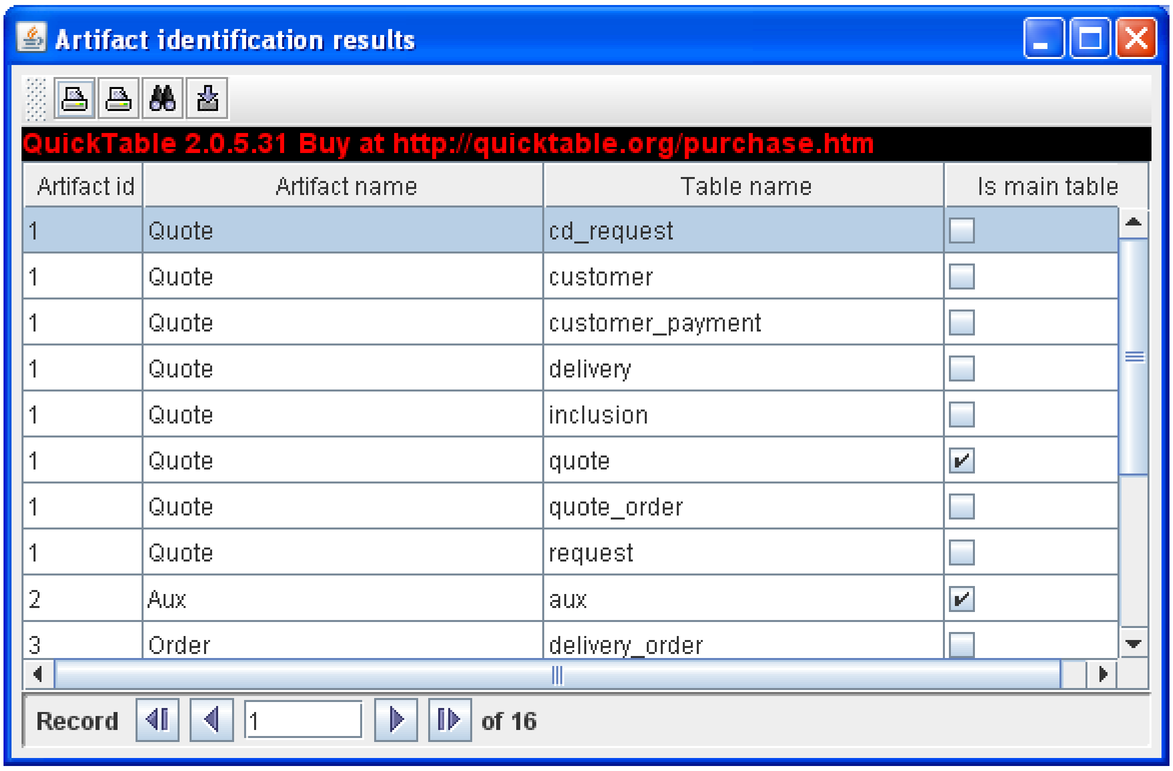
Number of clusters: 4
Expansion level: 0

Figure 43: Cluster tables parameters dialog

The dialog requests the following parameters:

- *Number of clusters* specifies the number of artifacts to identify.
- *Expansion level* specifies the maximum level of tables to add to the base clusters.

The currently identified artifacts can be viewed using the *View created artifacts* menu item. This opens the screen shown below (after a commercial dialog).



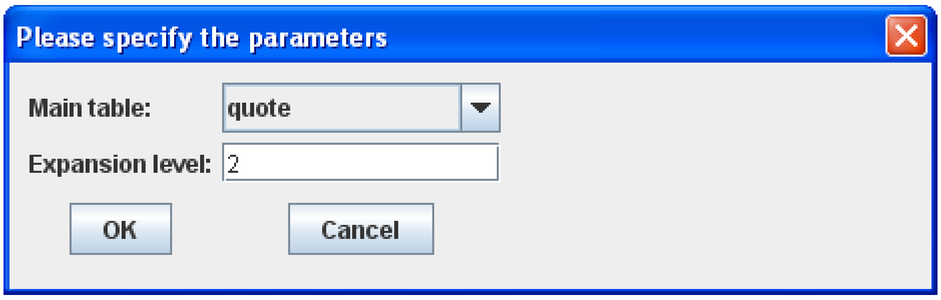
QuickTable 2.0.5.31 Buy at http://quicktable.org/purchase.htm

Artifact id	Artifact name	Table name	Is main table
1	Quote	cd_request	☐
1	Quote	customer	☐
1	Quote	customer_payment	☐
1	Quote	delivery	☐
1	Quote	inclusion	☐
1	Quote	quote	✓
1	Quote	quote_order	☐
1	Quote	request	☐
2	Aux	aux	✓
3	Order	delivery_order	☐

Record 1 of 16

Figure 44: Artifact identification results screen

An alternative way to create an artifact schema is provided by the *Create artifact* menu item. Here a main table can be specified manually, which will be expanded to an artifact schema using the allowed list of tables. First a dialog is shown in which the main table and maximum expansion level can be defined. The allowed list of tables must then be selected using the table selection dialog shown in Figure 36.



Main table: quote
Expansion level: 2

Figure 45: Create artifact parameters dialog

More detailed information about the implementation is available in the javadoc documentation of the *algorithms.DBSummarization* class and the *algorithms.clustering* and *model.clustering* package.

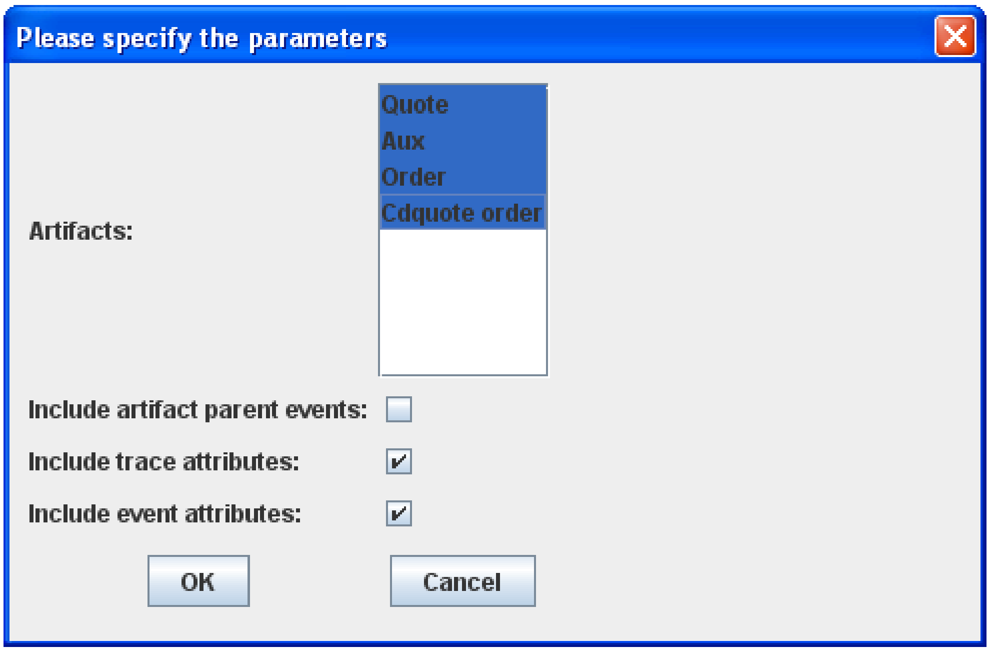
F.IV Artifact lifecycle identification

All techniques used in the artifact lifecycle identification step can be found in the *Artifact lifecycle identification* menu. These will only work if the artifact schemas were identified as specified in the previous section.

F.IV.I Create schema to log mapping

All menu items related to this subject can be found in the *Artifact lifecycle identification > Create schema-to-log mapping* menu.

The identification of event types and assignment of attributes is actived by the *Determine attribute & event selection* menu item. A dialog is shown to gather the required parameters.



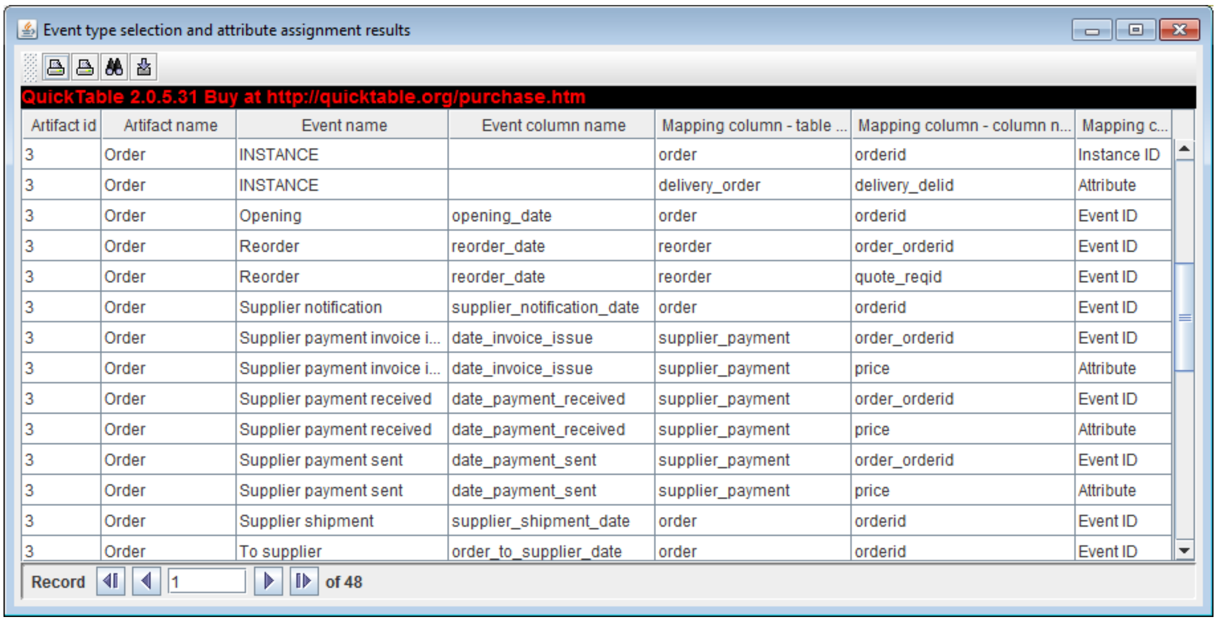
Artifacts: Quote, Aux, Order, Cdquote order
Include artifact parent events: ☐
Include trace attributes: ☑
Include event attributes: ☑

Figure 46: Attribute & event selection parameters dialog

The dialog requests the following parameters:

- *Artifacts* allows the selection of artifacts for which event types should be selected and attributes assigned.
- *Include artifact parent events* specifies if event types should be generated for timestamp columns that are part of a parent table of the main table.
- *Include trace attributes* specifies if non-event columns should be assigned as trace attributes. Note that the columns that would be trace attributes will never be assigned as event type attributes; this is just a way to limit the number of generated attributes.
- *Include event attributes* specifies if non-event columns should be assigned as event type attributes. Note that the columns that would be event type attributes will never be assigned as trace attributes; this is just a way to limit the number of generated attributes.

The result of the event type selection and attribute assignment can be shown using the *View selected attributes & event types* menu item. This opens the screen shown below (after a commercial dialog).

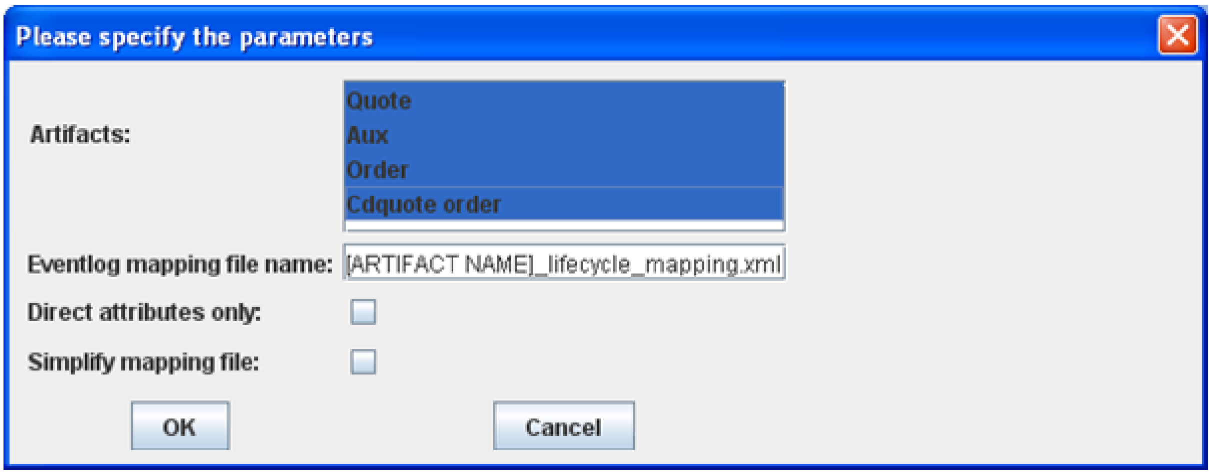

Event type selection and attribute assignment results
QuickTable 2.0.5.31 Buy at http://quicktable.org/purchase.htm

Artifact id	Artifact name	Event name	Event column name	Mapping column - table ...	Mapping column - column n...	Mapping c...
3	Order	INSTANCE		order	orderid	Instance ID
3	Order	INSTANCE		delivery_order	delivery_delid	Attribute
3	Order	Opening	opening_date	order	orderid	Event ID
3	Order	Reorder	reorder_date	reorder	order_orderid	Event ID
3	Order	Reorder	reorder_date	reorder	quote_reqid	Event ID
3	Order	Supplier notification	supplier_notification_date	order	orderid	Event ID
3	Order	Supplier payment invoice i...	date_invoice_issue	supplier_payment	order_orderid	Event ID
3	Order	Supplier payment invoice i...	date_invoice_issue	supplier_payment	price	Attribute
3	Order	Supplier payment received	date_payment_received	supplier_payment	order_orderid	Event ID
3	Order	Supplier payment received	date_payment_received	supplier_payment	price	Attribute
3	Order	Supplier payment sent	date_payment_sent	supplier_payment	order_orderid	Event ID
3	Order	Supplier payment sent	date_payment_sent	supplier_payment	price	Attribute
3	Order	Supplier shipment	supplier_shipment_date	order	orderid	Event ID
3	Order	To supplier	order_to_supplier_date	order	orderid	Event ID

Figure 47: Event type selection and attribute assignment results screen

Most columns in the result screen are self-explanatory. The *Event name* shows the name of the event type or "INSTANCE" for columns assigned to traces. The *Mapping column* shows information about columns assigned to event types or instances. The *Mapping column – type* shows how the column is related to the event type or instance: part of the instanceId, part of the eventId or assigned as an attribute.

The creation of the mapping is activated by the *Create event log mapping* menu item. A dialog is shown to gather the required parameters.


Please specify the parameters

Artifacts: Quote, Aux, Order, Cdquote order

Eventlog mapping file name: [ARTIFACT NAME]_lifecycle_mapping.xml

Direct attributes only: ☐

Simplify mapping file: ☐

Figure 48: Create event log mapping parameters dialog

The dialog requests the following parameters:

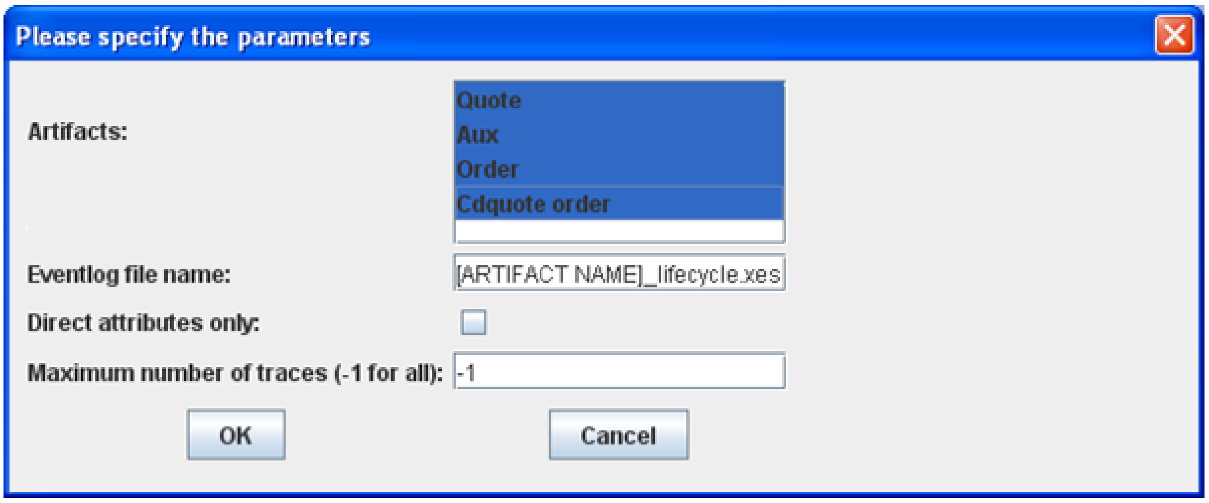
- *Artifacts* allows the selection of artifacts for which the schema-to-log mapping should be created.
- *Eventlog mapping file name* specifies the name of the resulting mapping file. "[ARTIFACT NAME]" will be replaced by the name of the artifact.
- *Direct attributes only* specifies if ListAttributes should be included in the mapping. This can be used to limit the number of queries executed during the event log generation step.
- *Simplify mapping file* can be used to create a more readable version of the mapping file. When this is selected various identifier fields will not be included, making the mapping more readable, but likely harder to process further.

More detailed information about the implementation is available in the javadoc documentation of the *algorithms.eventextraction.ArtifactEventLogSelection* and *algorithms.eventextraction.ArtifactEventLogMapper* classes and the *model.mapping* package.

F.IV.II Event log generation

All menu items related to this subject can be found in the *Artifact lifecycle identification > Generating traces* menu.

The generation of an event log can be activated using the *Create event log from datasource* menu item. A dialog is shown to gather the required parameters.


Please specify the parameters

Artifacts: Quote, Aux, Order, Cdquote order

Eventlog file name: [ARTIFACT NAME]_lifecycle.xes

Direct attributes only: ☐

Maximum number of traces (-1 for all): -1

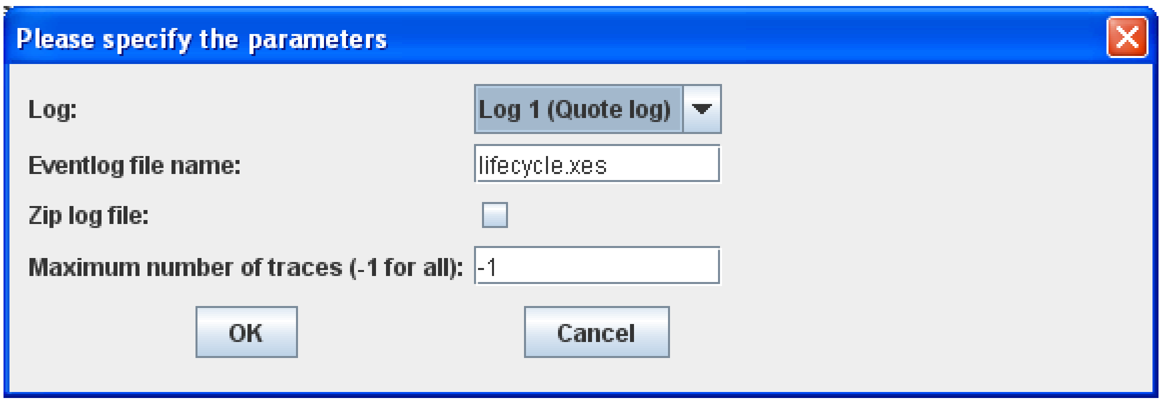
Figure 49: Create event log from datasource parameters dialog

The dialog requests the following parameters:

- *Artifacts* allows the selection of artifacts for which an event log should be created.
- *Eventlog file name* specifies the name of the resulting file. "[ARTIFACT NAME]" will be replaced by the name of the artifact.
- *Direct attributes only* specifies if ListAttributes should be included in the event log. This can be used to limit the number of queries executed.
- *Maximum number of traces* specifies the maximum number of traces that are to be included in the generated event log.

After the *OK* button is clicked the generation process will start. Due to OpenXES an active internet connection is required to create the event log.

If an event log was created before it can be created again from the cache database using the *Create Event Log from Cache* menu item. A dialog is shown to gather the required parameters.


Please specify the parameters

Log: Log 1 (Quote log)
Eventlog file name: lifecycle.xes
Zip log file: ☐
Maximum number of traces (-1 for all): -1

Figure 50: Create event log from cache parameters dialog

The dialog requests the following parameters:

- *Log* allows the selection of a cached event log.
- *Eventlog file name* specifies the name of the resulting file.
- *Zip log file* specifies if the created event log should be g-zipped.
- *Maximum number of traces* specifies the maximum number of traces that are to be included in the generated event log.

More detailed information about the implementation is available in the javadoc documentation of the *algorithms.eventextraction.ArtifactEventLogExtraction* and *algorithms.eventextraction.CacheToXesConverter* classes and the *model.xes* package.

F.V License(s)

A large number of libraries were used for this prototype that were all available under some form of open source license (e.g. GPL, LGPL, Apache, BSD). An overview of the used libraries and their licenses can be found in the "\Sources\packagelib" folder.

Aside from the libraries some parts of the code were also largely based on an external source with an open source license:

- Everything in the *org.apache.lucene* package was taken from Apache Lucene which is available under the Apache license.
- The original Gordian-HCA primary key discovery implementation was kindly donated by Ziawasch Abedjan under the Apache license. This concerns the classes with "Gordian" as part of their name in the *algorithms.pkdiscovery* package and the subpackages of this package with "gordian" or "histocount" in their name.
- The TANE functional dependency discovery implementation was taken from its original implementation which is available under the GPL license. This concerns the *TANE* and *FunctionalDependency* classes in the *algorithms.pkdiscovery*

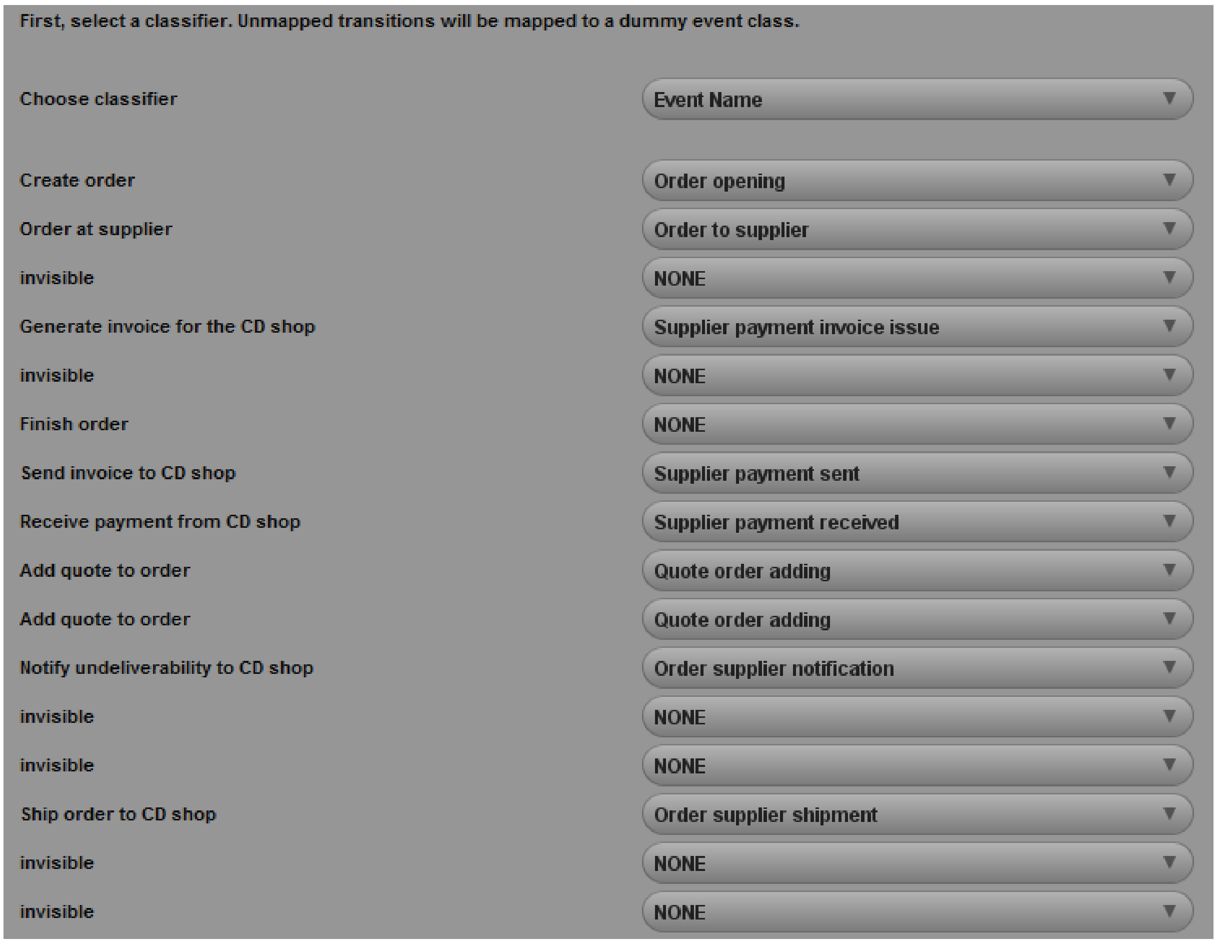
package, the *taneutils* subpackage and the *ComparableSet* class in the *utils.collections* package.

- The DBScan clustering algorithm was taken from WEKA which is available under the GPL license. This is the *DBScan* class in the *algorithms.clustering* package.
- The fast EMD computation was taken from its original Java implementation under the BSD license. This concerns everything in the *algorithms.emd* package.
- The *CacheToXesConverter* class in the *algorithms.eventextraction* package is based on the *CacheDBController* class of XESame. XESame is available under the Eclipse public license.

Appendix G: Lifecycle verification settings

G.I Order

The general mapping between the log and the original model is shown in Figure 51 below. The *finish order* event in the model was not present in the database and could thus not be mapped. Since the goal of the behavioural replay was to compare the automatically generated mapping with the best possible manual mapping the cost of the *finish order* event was set to 0 for the conformance check. Aside from this *reorder* event in the log could not be mapped to the original model, because it was not present there.



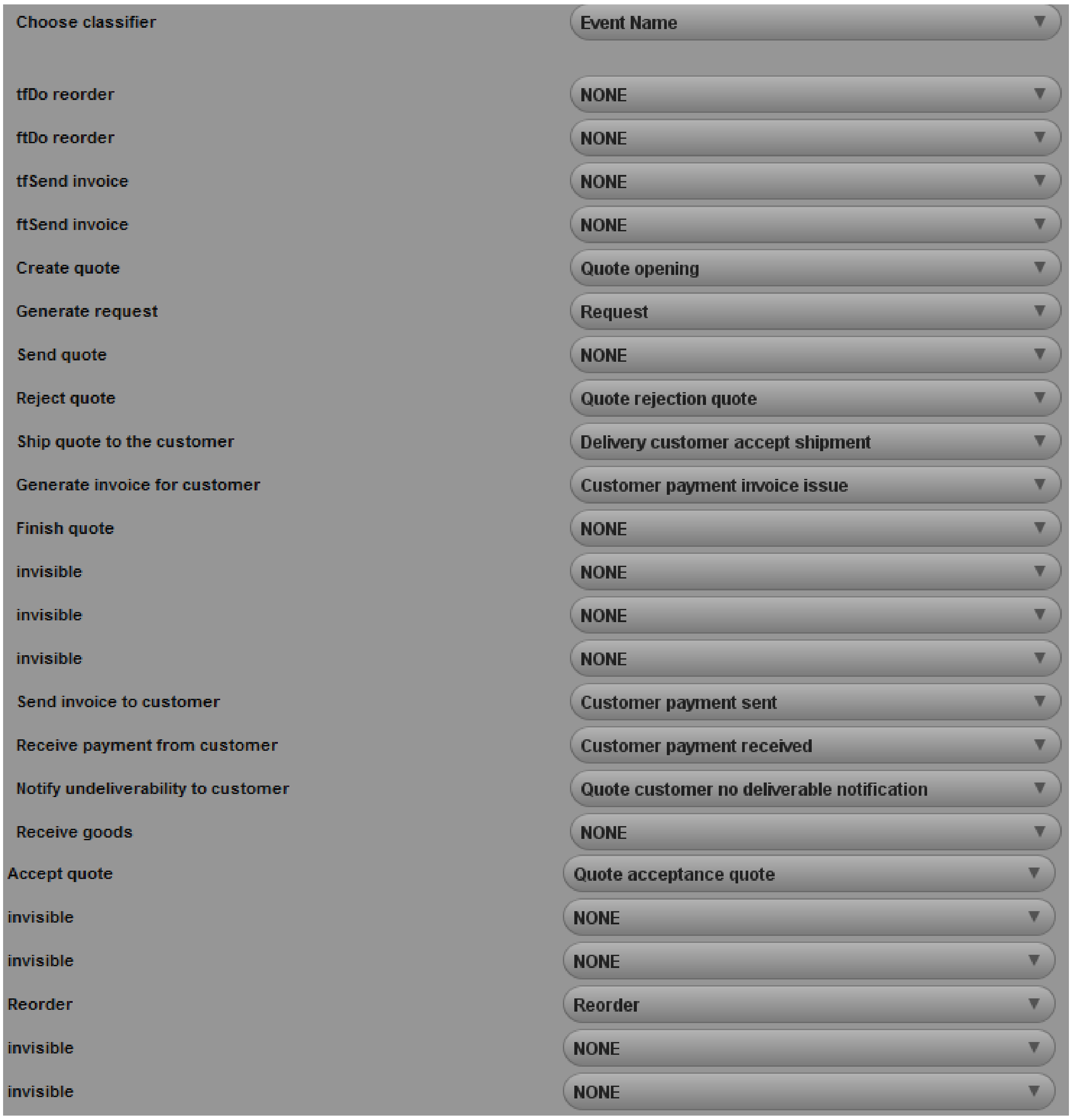
First, select a classifier. Unmapped transitions will be mapped to a dummy event class.

Choose classifier	Event Name
Create order	Order opening
Order at supplier	Order to supplier
invisible	NONE
Generate invoice for the CD shop	Supplier payment invoice issue
invisible	NONE
Finish order	NONE
Send invoice to CD shop	Supplier payment sent
Receive payment from CD shop	Supplier payment received
Add quote to order	Quote order adding
Add quote to order	Quote order adding
Notify undeliverability to CD shop	Order supplier notification
invisible	NONE
invisible	NONE
Ship order to CD shop	Order supplier shipment
invisible	NONE
invisible	NONE

Figure 51: Order original model - log mapping

G.II Quote

The general mapping between the log and the original model is shown in Figure 52 below. The *send quote*, *receive goods* and *finish quote* events in the original model could not be mapped to an event in the log, because this information was not represented in the database. Similar to what was described for the *order* log the cost for these events was set to 0 for the conformance check.



Choose classifier	Event Name ▼
tfDo reorder	NONE ▼
ftDo reorder	NONE ▼
tfSend invoice	NONE ▼
ftSend invoice	NONE ▼
Create quote	Quote opening ▼
Generate request	Request ▼
Send quote	NONE ▼
Reject quote	Quote rejection quote ▼
Ship quote to the customer	Delivery customer accept shipment ▼
Generate invoice for customer	Customer payment invoice issue ▼
Finish quote	NONE ▼
invisible	NONE ▼
invisible	NONE ▼
invisible	NONE ▼
Send invoice to customer	Customer payment sent ▼
Receive payment from customer	Customer payment received ▼
Notify undeliverability to customer	Quote customer no deliverable notification ▼
Receive goods	NONE ▼
Accept quote	Quote acceptance quote ▼
invisible	NONE ▼
invisible	NONE ▼
Reorder	Reorder ▼
invisible	NONE ▼
invisible	NONE ▼

Figure 52: Quote original model - log mapping