

MASTER

Event log extraction from SAP ECC 6.0

Piessens, D.A.M.

Award date:
2011

[Link to publication](#)

Disclaimer

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

General rights

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

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

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 14. Apr. 2026

Petitioner has added romanettes to pages (a) - (c). Otherwise, it leaves the original page numbering.

SAP
EXHIBIT 1014

(a)

**Event Log Extraction from
SAP ECC 6.0**

MASTER THESIS

D.A.M. Piessens



MASTER THESIS

Event Log Extraction from SAP ECC 6.0

Final Version

Author:

D.A.M. Piessens

Supervisors:

dr.ir. A.J. Mooij

dr.ir. G.I. Jojgov

dr. G.H.L. Fletcher

Eindhoven, April 2011

Abstract

Business processes form the heart of every organization; they can be seen as the blueprints through which all data flows. These business processes leave tracks in information systems like Enterprise Resource Planning, Supply Chain Management and Workflow Management Systems. Enterprise Resource Planning (ERP) systems are the most widely used ones; they control nearly anything that happens within a company. Most organizations keep records of various activities that have been carried out in these ERP systems for auditing purposes, but these are rarely used for analysis purposes and examined on a process level. From these recorded logs, valuable company information can be derived by looking for patterns in the tracks left behind. This technique is called process mining and focuses on discovering process models from event logs. The shift from data orientation to process orientation has demanded process mining solutions for ERP systems as well. Although many information systems produce logs, the information contained in these logs is not always suitable for process mining. A main step in performing process mining on such systems is therefore to properly construct an event log from the logged data.

In this thesis we propose a method that guides in extracting event logs from SAP ECC 6.0. The research is performed at Futura Process Intelligence; a company that delivers products and services in the area of process intelligence and monitoring, especially in the context of process mining. In the method we can identify two phases: a first phase in which we prepare and configure a repository for each SAP process, and a second phase where we actually perform the event log extraction. Within this method we introduce the notion of table-case mappings. These represent the case in an event log and they are computed automatically based on foreign keys that exist between tables in SAP. Additionally, we have developed and implemented a method to incrementally update a previously extracted event log with only the changes from the SAP system that were registered since the original event log was created. Our solution entailed the development of a supporting prototype as well, which is applied as a proof of concept on some case studies of important SAP processes. The developed application prototype guides the event log extraction for the configured processes in our repository.

Keywords: event log extraction, process mining, SAP ECC 6.0

Preface

The master thesis that lies in front of you concludes my academic studies at Eindhoven University of Technology. These started in September 2003 with a Bachelor study in Computer Science and Engineering, and was proceeded by a Master study Business Information Systems (BIS) in January 2009. The switch to BIS proved to be of added value through the addition of industrial engineering aspects; this, and the interest in the world of Business Process Management (BPM) has highly motivated me the last two years.

During my study I had the opportunity to develop my self in various ways. In 2006-2007 I was a full-time board member of the European Week Eindhoven, organizing this student conference with six fellow students was an incredible experience. Studying a semester abroad in Australia during my master has further raised my interest in BPM and process mining. I would especially like to thank Boudewijn van Dongen for his support in setting up the exchange semester with QUT and Moe Wynn for guiding me during my internship and motivating me to turn the internship research into an academic paper.

When looking for a master project, it was clear for me that I wanted to do something in the area of process mining. I again would like to thank Boudewijn for sharing his expertise and helping me in the initial phase of setting up this master project. Futura Process Intelligence, where the research project was conducted the past six months, has given me the freedom and opportunity to extend my knowledge of process mining and to take a look within their organization. The small size of the company only provided me with benefits; a lot of personal attention was given and practical experience was gained by daily discussing process mining projects. More specifically I would like to thank Peter van den Brand and Georgi Jojgov. Peter for his interest in my project and sharing his incredible knowledge of process mining, especially his experience with mining SAP. Georgi Jojgov became very important during my project; his daily guidance was very helpful, he identified future problems very quickly and showed to possess a lot of knowledge. Many thanks to Arjan Mooij as well, my supervisor at TU/e. He brought more academic depth in my project and guided my thesis to the next level with his remarks. Furthermore my thanks go out to George Fletcher for taking part in my evaluation committee and critically reviewing this document.

Furthermore I would like to thank my family for their support and interest in my studies. Especially my mother for stimulating me in my path to university. In my period at TU/e I would like to thank Latif, my college-buddy. We learned to work together in the last year of our Bachelor and kept on motivating eachother till the end of our studies. I am sure this thesis would not have been there earlier without him. Another person who plays an important role in my studies is Henriette. She showed me how to combine my student and social life and and sometimes made me exceed my expectations. Last but not least I would like to thank my girlfriend Laura for her ongoing love and (partly long distance) support during my master. Many thanks to all of my friends and other people that I cannot mention in detail as well. I would like to dedicate this thesis to all of you!

David Piessens
Eindhoven, April 2011

Contents

1	Introduction	1
1.1	Futura Process Intelligence	2
1.2	Research Scope and Goal	2
1.3	Research Method	3
1.4	Thesis Outline	4
2	Preliminaries	5
2.1	SAP	5
2.1.1	SAP ECC 6.0	6
2.1.2	Transactions	6
2.1.3	Common Processes in SAP ERP	7
2.2	Process Mining	9
2.3	Relational Databases	10
3	Related Work	13
3.1	TableFinder	13
3.2	Deloitte ERS	15
3.3	XES Mapper	16
3.4	Commercial Products	17
3.4.1	EVS ModelBuilder	18
3.4.2	ARIS Process Performance Manager	19
3.4.3	LiveModel	19
3.4.4	Fluxicon	19
3.4.5	SAP Solution Manager	20
3.5	Concluding Remarks	20
4	Extracting Data From SAP	21
4.1	Intermediate Documents	21
4.1.1	Principle	21
4.1.2	Evaluation	22
4.2	Database Approach	23
4.2.1	Obtaining Data	23
4.3	Conclusion	24

5	Extracting an Event Log	25
5.1	Project Decisions	25
5.1.1	Determining Scope and Goal	25
5.1.2	Determining Focus	26
5.2	Procedure	26
5.3	Preparation Phase	27
5.3.1	Determining Activities	27
5.3.2	Mapping out the detection of Events	30
5.3.3	Selecting Attributes	33
5.4	Extraction Phase	34
5.4.1	Selecting Activities to Extract	34
5.4.2	Selecting the Case	34
5.4.3	Constructing the Event log	35
5.5	Conclusion	35
6	Case Determination	37
6.1	Table-Case Mapping	38
6.1.1	Base Tables	38
6.1.2	Foreign Key Relations	39
6.1.3	Computing Table-Case Mappings	41
6.2	Divergence and Convergence	45
6.2.1	Divergence	46
6.2.2	Convergence	47
6.3	Ongoing Research	48
6.3.1	Artifact-Centric Process Models	48
6.3.2	Possibilities for SAP	49
6.4	Conclusion	50
7	Incremental Updates	51
7.1	Overview	51
7.1.1	Assumptions	53
7.1.2	Decisions	53
7.1.3	Exploration	53
7.2	Update Procedure	54
7.2.1	Update Database	55
7.2.2	Select Previously Extracted Event Log	55
7.2.3	Update Event Log	55
7.3	Conclusion	56
8	Prototype Implementation	57
8.1	Overview	57
8.1.1	Preparation Phase	58
8.1.2	External Interfaces	64
8.2	Incremental Updates	66
8.2.1	Overview	66
8.2.2	Prototype Extensions	66
8.3	Technical Structure	69

8.3.1	Implementation Details	69
8.3.2	Class Diagram	69
8.4	Graphical User Interface	70
8.4.1	Selecting Activities	71
8.4.2	Computing Table-Case Mappings	71
8.4.3	Extracting the Event Log	72
8.4.4	Extraction Results	74
8.4.5	Updating the Database	75
8.4.6	Updating the Event Log	76
8.5	Incremental Update Improvements	77
8.6	Conclusion	78
9	Case Studies	79
9.1	Purchase To Pay	79
9.1.1	Activities	79
9.1.2	Table Characteristics	80
9.1.3	Purchase Order Line Item Level	80
9.1.4	Purchasing Document Level	85
9.1.5	Comparison	86
9.1.6	Purchase Requisition Level	88
9.1.7	Incremental Update of an Event Log	90
9.2	Order To Cash	91
9.2.1	Activities	91
9.2.2	Table Characteristics	91
9.2.3	Sales Order Item Level	92
9.3	Conclusion	95
10	Conclusions	97
10.1	Future Work	98
A	Glossary	103
B	Downloading Data from SAP	105

Chapter 1

Introduction

Business processes form the heart of every organization. From small companies to large multinationals, a number of business processes can always be identified in the organization and their information systems. These business processes leave tracks in information systems like Enterprise Resource Planning, Supply Chain Management and Workflow Management Systems. *Enterprise Resource Planning (ERP)* systems are the most widely used ones, they control nearly anything that happens within a company, be it finance, human resources, customer relationship management or supply chain management. Most organizations keep records of various activities that have been carried out in these ERP systems for auditing purposes, but these are rarely used for analysis purposes and examined on a process level.

From these recorded logs, valuable company information can be derived by looking for patterns in the tracks left behind. This technique is called *process mining* and focuses on discovering process models from *event logs*. Event logs are a more structured form of logs, and contain information about cases and the events that are executed. Ideally the involved information systems are process-aware [7]; workflow management systems are typical examples of such systems. The shift from data orientation to process orientation has however led to the fact that process mining solutions are also demanded for non process-aware information systems. These data-oriented systems, like most ERP systems, are often of vital importance to a company and need to be analyzed on a process level as well. Future information systems that anticipate the value of process mining may facilitate the extraction of event logs for these systems, but for the moment this step requires considerable manual effort by the event log extractor.

The ERP system on which the research is done is SAP ECC 6.0, a software package widely used across the world. Several important processes can be identified within SAP (e.g. Order to Cash, Purchase to Pay); event logs for these processes are not readily available, but event related information is stored in the SAP database. SAP is often installed throughout various layers of a company, and few users, if any, have a clear and complete view of the overall process.

A data-centric system like SAP was not designed to be analyzed on a process level. If it is possible for a company to translate their SAP data into process models, benefits could be gained by becoming aware of the actual data flow. In order to do that, events need to be derived from data spread across various tables in SAP's database. Before we can apply

process mining techniques, we first have to create an event log from this data. Since event logs are the (main) input to perform process mining, we can summarize the problem statement as follows:

Problem Statement: *SAP ECC 6.0 does not provide suitable logs for process mining.*

In this chapter we define the above mentioned problem in detail and start off by providing more information about the company where this graduation project is performed: *Futura Process Intelligence* (Section 1.1). The scope and goal of the research are set in Section 1.2, and Section 1.3 presents the research method. In Section 1.4 we conclude by outlining the structure of this thesis.

1.1 Futura Process Intelligence

With its roots in Eindhoven University of Technology, Futura Process Intelligence delivers products and services in the area of Process Intelligence and Monitoring. They are particularly focused on the development of professional process mining software for commercial purposes. The connection with Eindhoven University of Technology, a pioneer in the field of process mining, provides them the opportunity to be the first to apply new process mining techniques and pick in on existing research.

Started up in the fall of 2006, Futura is still a relatively new company and the market is still reluctant towards this new way of analysing processes. However, more and more companies acknowledge the added value of process mining and consult Futura for an in-depth analysis of their processes. Based on scientific research on process mining, Futura has built Reflect. Futura Reflect is a Process Intelligence and Process Mining application that supports automatic process discovery, process animation, performance analysis and social network discovery. Reflect is being offered as Software as a Service (SaaS). They offer a range of consulting services in these areas as well to aid companies in setting up and applying process mining within their company. For example, Futura offers a 14 Day Challenge¹, where, in a very short period of time, they analyse a mutually agreed-on business process.

In 2009, Futura was elected as one of the ‘Cool Vendors in Business Process Management’ by Gartner [9]. Gartner specifically praises Futura’s work on automated business process discovery (ABPD): “*Factors that differentiate Futura from many other offerings in the field of BPM include its strong focus on staying ahead of the curve by innovating and the highly intuitive way it provides insight into the historical execution of a process using a novel process animation technique*”.

1.2 Research Scope and Goal

Futura Process Intelligence’s area of expertise thus lays in process mining. A re-occurring problem within the company these days is how to extract event logs for SAP processes. Futura already has experience with mining some of these SAP processes, but this knowledge is rather small and continues to pose them problems since the solutions are rather limited and *process-specific*.

¹<http://www.14daychallenge.nl>

We can summarize the project goal as follows:

Project Goal: *Create a method to extract events logs from SAP ECC 6.0 and build an application prototype that supports this.*

Ideally, this method should be applicable to all business processes that can be implemented in SAP. Figure 1.1 visualizes the project goal; we focus on the entire event log extraction procedure, from acquiring data from SAP to constructing the event log in Futura’s CSV format. Having obtained these event logs, process mining could be applied to discover the ‘real’ process, analyse it, compare it with how persons normally perceive the process and try to improve it. This is however outside the scope of the project, the focus in this project only lays on the actual extraction of the event log from SAP ECC 6.0.



Figure 1.1: Project Goal

1.3 Research Method

To achieve the project’s goal and solve the problem statement, we set out a research method that can be divided into various smaller steps. Below we enumerate the points that need to be tackled:

1. Gain insight in how and where data is logged within SAP.
2. Research how this data relates to an SAP business process.
3. Create a method to determine the relations between logged data.
4. Create a method to extract this logged data from SAP.
5. Determine ways to group the data in terms of cases.
6. Transform the extracted data to an event log.
7. Investigate how to deal with updated data records.

The results of these steps should support us in creating a method that guides in extracting event logs from SAP. Additionally we address the question of how to deal with updated data, something new that distinguishes this research from previous research. Ideally, and this is where the real challenge lies, this results in a method to incrementally update a previously extracted event log with only the changes from the SAP system that were registered since the original event log was created. All this is supported by a prototype, which as a proof of concept is applied on some case studies of important SAP processes.

The following are expected outcomes of the project:

- A method to extract event logs from SAP ECC 6.0
- A method to determine possible cases for a given process
- A method to incrementally update a previously extracted event log
- A supporting prototype

1.4 Thesis Outline

The outline of this thesis is presented below and is driven by the research method; we have the following chapters:

Chapter 2	Introduces some preliminary concepts that are used throughout this thesis.
Chapter 3	Presents the results of a literature and software survey to find gaps in the literature and specific points that can be improved or researched.
Chapter 4	Discusses and evaluates two approaches that have been investigated to retrieve data from SAP's database.
Chapter 5	Presents the main procedure to extract event logs from SAP ECC 6.0.
Chapter 6	Presents a method to propose cases for a given set of activities.
Chapter 7	Investigates how to deal with updated data records and presents a method to (incrementally) update a previously extracted event log.
Chapter 8	Presents the application prototype that supports the event log extraction process.
Chapter 9	Presents two case studies that test the prototype and validate the approach.
Chapter 10	Concludes by evaluating the entire approach and arguing whether we achieved the goal; future work is discussed here as well.
Appendix A	Presents a glossary with important terms used throughout this thesis.

Chapter 2

Preliminaries

This chapter introduces preliminary concepts used throughout this thesis. Section 2.1 introduces *SAP*: the company, the ERP system, the notion of transactions, and some common SAP business processes. The principle of *process mining* is explained in Section 2.2, where we focus the attention on event logs. Section 2.3 briefly introduces some *relational database* concepts that are extensively used throughout this thesis: tables, primary keys and foreign keys.

2.1 SAP

SAP, short for *Systemanalyse und Programmentwicklung* (System Analysis and Program development), was founded in 1972 as SAP AG by five former IBM engineers. They are the worldwide number one company that specializes in enterprise software and the world's third-largest independent software provider overall. The solutions they provide can be applied from small to mid-size companies as well as large international organizations. They are headquartered in Walldorf, Germany and have regional offices all around the world. They are best known for their *Enterprise Resource Planning* product and their consultancy branch which implements their products and provides training to end users. According to SAP's annual report of 2009 [19], SAP AG has more than 95.000 customers in over 120 countries and employ more than 47,500 people at locations in more than 50 countries worldwide.

Nowadays, SAP is moving to an Enterprise Service-Oriented Architecture (E-SOA). E-SOA allows them to reuse software components and not rely as much on in-house ERP hardware technologies, which makes it more attractive for small and mid-sized companies. All new SAP products are based on this E-SOA technology platform (i.e. SAP NetWeaver). This provides the technical foundation for SAP applications and guidance to support companies in creating their own SOA solutions comprising both SAP and non-SAP solutions. You can say that it offers an enterprise wide blueprint for business process improvement.

The version of SAP ERP we use in this master project, SAP ECC 6.0, is presented in Section 2.1.1. Section 2.1.2 introduces the concept of transactions, the key in using SAP ECC 6.0. Two common business processes that are implemented in SAP ERP, the Purchase to Pay and Order to Cash process, are outlined in Section 2.1.3.

2.1.1 SAP ECC 6.0

During the course of years, several versions of the SAP Enterprise Resource Planning (ERP) application have been released. The most well known, and still widely implemented version is SAP R/3. Launched in July 1992, it consists of various applications on top of SAP Basis, SAP's set of middleware programs and tools. Changes in the industry led to the development of a more complete package: mySAP ERP. Launched in 2003, the first edition of *mySAP* bundled previously separate products as SAP R/3 Enterprise, SAP Strategic Enterprise Management (SEM) and extension sets.

An architecture overhaul took place with the introduction of mySAP ERP Edition 2004. ERP Central Component (SAP ECC) became the successor of R/3 Enterprise and was merged with SAP Business Warehouse (SAP's Data Warehouse), SEM and much more which allowed users to run all these SAP solutions under one instance. This architectural change has been made to support an enterprise services architecture to help customers transitioning to an SOA. Traditionally, in each SAP ERP implementation the typical functions are arranged into distinct *functional modules*. The most popular are Finance and Controlling (**FI/CO**), Human Resources (**HR**), Materials Management (**MM**), Sales and Distribution (**SD**) and Production Planning (**PP**). Due to the size and complexity of these **modules**, SAP consultants are often specialised in only one of these modules.

In this graduation project, an installation of SAP ECC 6.0 is used for testing purposes, more specifically **SAP IDES ECC 6.0**. IDES, the Internet Demonstration and Evaluation System, represents a model company and consists of an international group with subsidiaries in several countries. Application data (designed to reflect real-life business requirements) for various business scenarios that can be run in the SAP system is stored in an underlying **relational database**.

2.1.2 Transactions

Users can start tasks in SAP by performing **transactions**. SAP transactions can either be executed directly by entering the correct transaction code in the SAP menu, or indirectly by selecting the corresponding task description from the SAP Easy Access menu. Both these methods result in a call to the corresponding ABAP program for the transaction; so transactions are simply shortcuts to execute ABAP programs. ABAP (Advanced Business Application Programming) is SAP's developed and used programming language to write programs for SAP. For example, transaction code *ME51N* lets you perform the task *Create Purchase Requisition*, while transaction *F-28* handles an incoming payment of a customer. Some transactions are just there to consult information and not to perform changes to stored data, like *SE84*, which gives access to the *Repository Information System*, or *SW01* which opens the *Business Object Browser*.

In total there are about 106.000 transactions in SAP ECC6.0. Finding the desired transaction code for a specific task is often challenging since descriptions are often cryptic or difficult to find.

2.1.3 Common Processes in SAP ERP

With decades of experience, SAP has created a set of **best practices** that companies can use as a reference model to construct their own business processes. These best practices are often tailored further by companies themselves and form a good starting point for companies to implement SAP ERP. Information, excluding process models, about the best practices can be found online at the SAP website (like the steps that are involved and how they can be executed). With the help of these best practices it is possible to get an idea of how a process should be implemented in SAP and how it looks like.

This section delves deeper into two important processes in SAP for which also a best practice exists. First of all, the **Purchase to Pay (PTP)** process. This process demonstrates the *entire process chain in a typical procurement cycle*. The second process, **Order to Cash (OTC)**, supports the *process chain for a typical sales process with a customer*. Both processes contain several phases. If a certain SAP process is not known beforehand, a best practice for such a process provides a good first insight in the various phases.

1. Purchase to Pay

The Purchase to Pay process (or Procure to Pay, PTP) focuses on *procurement of trading goods*. It is one of the most common processes and often the key process within a company. Several variations of this process exist; the SAP best practice *Procure To Pay for a Wholesale Distributor*¹ consists of the following steps:

- Source Determination
- Vendor Selection and Comparison of Quotations
- Determination of Requirements
- Purchase Order Processing
- Purchase Order Follow-Up
 - Goods Receiving (with quality management) and Inventory Management
 - Invoice Verification
 - Payment Execution

The above steps are more general descriptions of actions that should be done in the PTP process. In Figure 2.1, these steps are translated into SAP terminology and the PTP process is depicted as a cycle (procurement cycle). In this simplified cycle the Materials Management (MM) and Financial (FI) module are involved. Purchase Requisition, Purchase Order, Notify Vendor and Vendor Shipment are done through the MM module, while Goods Receipt, Invoice Receipt and Payment to Vendor belong to the FI Module.

Besides the actions given in Figure 2.1 and the list above, many more actions exist in this process. For example, *deleting a Purchase Requisition, changing a Purchase Order, blocking a Purchase Order, blocking a Payment* etc. All these *sub actions* can be retrieved as well and are considered in this thesis. They can provide additional information about the process; note that (sequences of) actions that deviate from the main flow (i.e. outliers) often turn out to be the most interesting ones. Furthermore, companies implement the procurement process

¹http://help.sap.com/bp_bblibrary/500/html/W30_EN_DE.htm

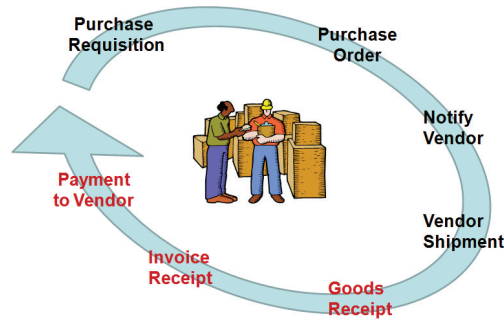


Figure 2.1: Procurement Cycle

as they like, and variations between PTP processes may exist. The PTP process is addressed several times in the remainder of this thesis and is analyzed further in a case study for the IDES system in Section 9.1.

2. Order to Cash

The Order to Cash (OTC) business process covers *standard Sales Order processing*, that is, from *creating the Sales Order*, to *Delivery* to *Billing*. The OTC process is a SAP best practice as well, *Order To Cash for a Wholesale Distributor*² consists of the following steps:

- Quotation
- Sales order with quotation reference
- Delivery
 - Picking with automatic transfer order creation and confirmation
 - Picking with manual transfer order creation
 - Confirmation
 - Packing
 - Posting goods issue
- Billing
- Payment by customer

The above mentioned steps provide a first insight in the OTC process, a translation of these concepts to SAP terminology is given in Figure 2.2, where the OTC process is presented as a sales order cycle. The FI, SD and Warehouse Management (WM) modules are used by the process. SD handles everything related to creation and changing of a Sales Order. Warehouse Management is more related to the goods in the Sales Order itself. It assists in processing all goods movements and in maintaining current stock inventories in the warehouse, like processing goods receipts, goods issues and stock transfers (transfer order). The FI module is of course used to handle incoming payments of a customer.

The Sales to Order process is mined from the IDES system as well, an in depth case study on the extraction of an event log for the OTC process can be found in Section 9.2.

²http://help.sap.com/bp_bblibrary/500/html/W40_EN.DE.htm

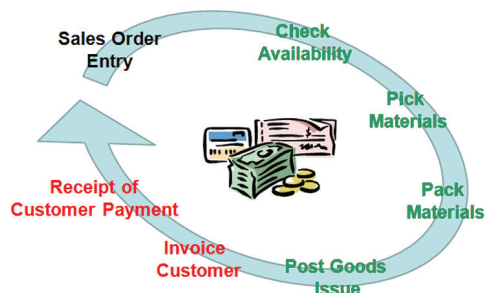


Figure 2.2: Sales Order Cycle

2.2 Process Mining

Process mining is a technology that uses event logs (i.e. recorded actual behaviors) to analyse executable business processes or workflows [1]. These techniques provide insight into control flow dependencies, data usage, resource utilization and various performance related statistics. This is a valuable outcome in its own right, since such dynamically captured information can alert us to problems with the process definition, such as ‘hotspots’ or bottlenecks that cannot be identified by mere inspection of the static model alone.

One of the goals of process mining (*discovery*) is to extract process models from event logs. These process models can only be discovered if the system, e.g. SAP ECC 6.0, is recording the actual behavior of the system. Event logs contain **events**; events are occurrences of **activities** in a certain process for a certain **case**. Each event is thus an instance of a certain activity. A case is an object that passes through a process. Examples are persons, purchase orders, complaints etc. When a new case is created in such a process, a new instance of the process is generated which is called a **process instance**. The trace of events that are executed for a specific case should all refer to the same process instance in the event log. The order of events is defined by a date and time (*timestamp*) attribute of the event, and determines the sequence in which activities occurred. Another common attribute is the *resource* that executed the event, which can be a user of the system, the system itself or an external system. Many other attributes can be stored within the event log, attributes that contain specific information about the case/event (e.g. vendor, price, amount, quantity etc.).

Process mining closes the gap between the limited knowledge process owners have about their company’s processes and the process as it is actually executed (the AS-IS process). It completes the process modeling loop by allowing the **discovery**, analysis (**conformance**) and **extension** of process models from event logs (Figure 2.3). In (1) Discovery, based on an event log, *a process model is automatically constructed*. For example, the genetic miner from Futura Reflect is constructed around a genetic algorithm that can mine models with all common structural constructs that can be found in process models [16]. (2) Conformance checking of process models is used to check if *reality conforms to the model*. It detects, locates, explains and measures these conformance deviations. In the third class, (3) Extension, we *enrich a process model with data from the accompanied event log*. An example is the extension of a process model with performance data. Futura Reflect provides this by giving the possibility to project performance metrics on the process models.

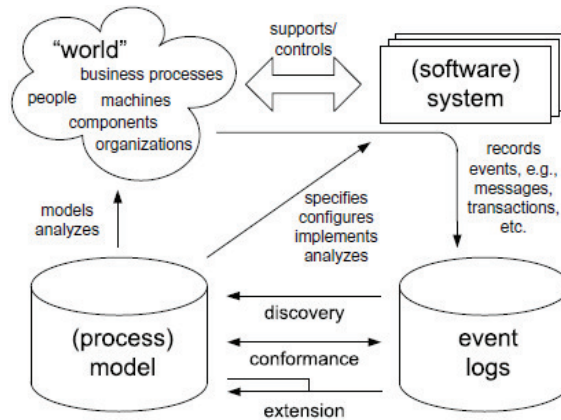


Figure 2.3: Three Classes of Process Mining Techniques

On the research side of process mining there exists a generic open-source framework, ProM, in which various process mining algorithms have been implemented [6]. The framework provides researchers an extensive base to implement new algorithms in the form of plug-ins. Looking from a commercial perspective, the popularity of process mining is still lacking behind other *business intelligence* solutions. Futura Reflect is the most commercially used process mining framework; however, the added value of process mining is acknowledged more than ever and it will not take long before more companies engage the competition and enter the field of process mining.

2.3 Relational Databases

The relational database model uses a collection of tables to represent both data and the relationships among those data [21]. The relational data model is the most widely used data model; a vast majority of current database systems are based on the relational model. As mentioned earlier, SAP ECC 6.0 stores its data in an underlying relational database as well. In the upcoming sections we introduce some more preliminary database concepts which will be useful later on.

Tables

Each *table* in a relational database is a set of data elements that are organized in a tabular format. The vertical columns are identified by their unique column name and have an accompanied data format (e.g. text or integer). The number of columns is specified for each individual table, but each table can have any number of rows. Each row is identified by the values appearing in a particular column subset (set of fields), which is referred to as the *primary key*.

Primary Keys

The *primary key* of a relational table uniquely identifies each record in that table. It is composed of a set of attributes in that table; for each value of the primary key we have at

most one record in the table. It can for example be one attribute that is guaranteed to be unique (e.g. social security number in a table with no more than one record per person).

Foreign Keys

A *foreign key*, often a combination of fields, links two tables T1 and T2 by assigning (a) field(s) of T1 to the primary key field(s) of T2. Table T1 is called the **foreign key table** (dependent table) and table T2 the **check table** (reference table). Each field of the foreign key table corresponds to a key field of the check table, this field is called the **foreign key field**. The combination of check table fields form the **primary key** of the check table. Different cardinalities may exist for foreign keys which express how the tables are exactly related (e.g. one-to-many, many-to-one). Thus, *one record of the foreign key table uniquely identifies at most one record of the check table using the entries in the foreign key fields*.

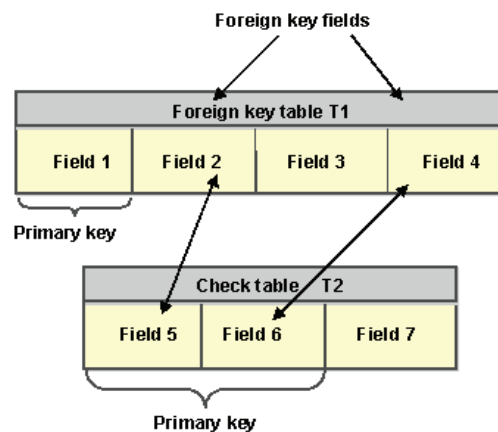


Figure 2.4: Foreign Keys

Chapter 3

Related Work

The growing popularity of process mining and the continuing presence of SAP in the corporate world has asked for process mining solutions for SAP. Section 3.1 presents and discusses the work of the pioneer in the field of process mining in SAP, Martijn van Giessel. Another Master's thesis is presented in Section 3.2. This considers Process Mining in an audit approach and includes a case study on SAP. A third (more recent) Master thesis performed at Eindhoven University of Technology is discussed in Section 3.3. Joos Buijs proposed and implemented an approach to map data sources in a generic way to an event log. Although his thesis does not target SAP as the main source of data, it does present a case study in which his implementation is applied to an SAP procurement process. Furthermore, Section 3.4 introduces several tools and companies that create process mining software or that apply similar business process intelligence techniques. We compare each approach in the following sections with the goals that are introduced in Chapter 1. We take note of interesting ideas and list the limitations each approach/software product has. There are four points we specifically focus on:

1. Genericity of the approach
2. Level of automation
3. Determination of cases
4. Updating of event logs

3.1 TableFinder

Process Mining is a relatively new concept. One of the first to investigate the applicability of Process Mining on SAP was Martijn van Giessel in 2004 [10]. In his Master thesis, *Process Mining in SAP R/3*, the central question is how the concept of process mining can be applied in an SAP R/3 environment. He splits his research into three parts:

1. How to find the relevant tables from which data must be extracted?
2. How to find the relationships between the relevant tables?
3. How to find a task description (event name) linked to a document number (document identifier)?

As a basis for his research he uses the SAP reference model [5]. This model consists of four views, which together represent business processes. One of the views, the *object/data model*,

contains all business objects that are needed for executing a task in a business process, and is thus the most important for process mining. The *business objects* are again related to tables, and therefore *form the key to finding the relevant tables*. In his study he uses the information from the reference model to extract information. First, the application component for the concerned process needs to be determined (e.g. Financial Accounting); then, the business objects that are involved should be identified (business objects belong to a specific application component). Van Giessel then uses TableFinder, an application developed in Visual Basic for Applications, to determine the tables that are related to those business objects. The input for the application consists of SAP R/3 reports and contains information about business objects, entities, tables and relationships of a given data model. The next and most difficult step is to determine the document flow. This is done through MS Excel by sorting and linking tables, a quite laborious and manual task. As a last step when having acquired the document flow of the process, an XML event log is constructed *by hand*.

Van Giessel's work proposes indeed a method to apply process mining techniques in SAP R/3, however several shortcomings can be identified in his work.

- Determining the business objects that are related to a specific SAP process is time consuming. In-depth SAP knowledge about a process is needed to be able to determine the involved business objects.
- Retrieving the document flow manually through MS Excel is very laborious for a large number of events.
- Each SAP R/3 installation is tailored to the client's needs. Because van Giessel's approach is heavily dependent of the SAP reference model, if a business process deviates from the standard processes implemented in this model, an inaccurate view of the business process may be acquired.
- The concept of *Convergence* and *Divergence*, further explained in Section 6.2, is not addressed.
- The event log is constructed by hand. For large amounts of data, which is normal in SAP, this creates problems.

If we generalize bullet point number three, van Giessel's method to automatically determine the relevant tables returns **all** tables for a given *Application Area* (e.g. *Purchasing*). This is often more than needed for a process that (partially) resides in this application area. Thus, the determined tables are not (directly) related to the activities that actually occur.

This being the first research done in this area, the method indeed lays a basis for process mining in SAP R/3 and acknowledges that SAP does not produce suitable event logs for process mining. The SAP Reference Model proved to be very useful to gain insight in the way SAP R/3 logs its information; however, van Giessel's method is not generic enough to build on for my own research. Additionally, some years after van Giessel's thesis, some mistakes were detected in the SAP reference models. In Mendling et al. [17], the authors investigated a model collection of about 600 EPC process models that are part of the SAP Reference Model. It turned out that at least 34 of these EPCs contain errors. Because of this, the fact that the models are outdated and that companies more and more deviate from these models, the SAP

reference models are not included anymore in newer versions of SAP. Other products, like the SAP Solution Manager and LiveModel discussed in Section 3.4, provide and maintain reference models for companies to use as a starting template. They are kept up to date and form the connection between the *workflow view* of a process and *SAP*. However, these templates are not publicly available and differ per company. The best practices mentioned in Section 2.1.3 form a good replacement for this, although they do not provide models, they can be used as a source to gain insight in the various processes that can be implemented through SAP.

Van Giessel's method is entirely focused on extracting data from the SAP Relational database. He accurately describes how to extract data from the database; the appendices in particular give a lot of practical information on how tables are related and how all the information can be accessed in SAP through transaction codes. However, the identified limitations stress the importance of creating a new approach for determining the case of a business process, (automatically) constructing the event log and updating the event log incrementally.

3.2 Deloitte ERS

In [20], Segers researched the applicability of process mining in the audit approach. This study on Deloitte Enterprise Risk Services concerns a Master's thesis performed in 2007 at the Industrial Engineering and Innovation Sciences faculty of TU/e. It uses ProM and the ProM import framework to support the analysis. By using a model-driven approach, a model for using process mining in a general business cycle was developed. This encompassed specifying a requirements model for applying process mining for testing application controls in the expenditure cycle, and a model for applying process mining in the SAP R/3 environment. Segers again proves the technical feasibility of process mining in an ERP package, and indicated that it is not that straightforward. He is one of the first to pinpoint the problems with *convergence and divergence*, and mentions the laborious work that is accompanied with extracting an event log where such issues occur. Setting up an extraction and conversion mechanism in order to create an event log is proven to be very dependent on the data structure.

The information about auditing and business models developed is quite extensive and not relevant for my project. The most interesting part of Segers' work concerns his study on the PTP process. This however does not contain detailed information about the actual event log construction and merely presents us new information about the PTP process. The creation of the event log is done with help of the ProM import framework and is further analysed with ProM 5. Extraction of the event log is performed on a very small scale and again requires a lot of manual work.

Concluding, Segers proposes that developing extraction procedures for specific SAP cycles (SAP business processes) would be very beneficial since mining an SAP process is largely dependent on the way data is stored in tables. One of the goals of my project conforms to this proposal: build a repository to smoothen the event log extraction for previously extracted processes. This means that eventually, for each SAP process, a method should be readily available to extract the log.

3.3 XES Mapper

In a more recent study from 2010, *Mapping Data Sources to XES in a Generic Way* [4], Joos Buijs performed research on how to extract event logs from various data sources. His thesis first discusses all the various aspects that should be considered when defining a conversion for data to an event log. This includes trace-, event- and attribute selection, as well as important project decisions that should be made beforehand. Another large portion of his chapter on aspects is devoted to the concept of convergence and divergence, a notion frequently observed in SAP.

Defining a conversion definition is the main principle of Buijs' work. A framework to store aspects of such a conversion is developed. In this framework, the extraction of traces and events, as well as their attributes, can be defined. Buijs developed an application prototype, called *XES Mapper*, that uses this conversion framework. The application guides the definition of a conversion, following three execution phases as depicted in Figure 3.1.

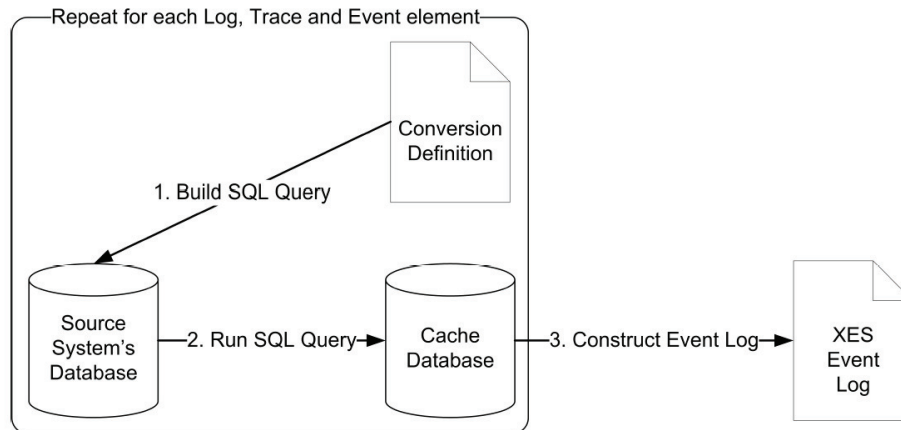


Figure 3.1: The three execution phases of the implementation

It is assumed that the data is available in the form of a relational database. Having this data, the first step is to create an SQL query from the conversion definition for each log, trace and event instance. The second step is to run each of these queries on the source system's database. The results of these queries are to be stored in an intermediate database. The third step is to convert this intermediate database to an XES event log for ProM.

Applying Buijs' application on SAP processes is still very laborious. We acknowledge the following limitations:

- The developed application assumes that a relational database containing data is available. In the SAP case study presented in section 6.1 of Buijs' work, this data is provided by LaQuSo, the laboratory for Quality Software, a joint initiative of Eindhoven University of Technology and Radboud University Nijmegen. All relations between the tables were set, and information about tables was available. In my thesis, this is not assumed to be known. Therefore, extracting the data from SAP is important to consider as well.

- Creating the conversion definition requires a lot of domain knowledge and SQL querying. Understanding the system and the process you are trying to mine is therefore very important.
- The frequently recurring problem of *Convergence* and *Divergence* is discussed, but no solution is proposed or given.
- How to deal with updated data records and tables is not addressed.

Buijs' work addressed several issues and aspects which also should be considered during my thesis. The research method is well-established, but not specifically targeted on SAP processes. A case study is presented, but this only shows the creation of a log with SAP data already available in the form of a relational database. Although our data in SAP is also available in the form of a relational database, Buijs' does not discuss how to detect events from these tables. An important aspect in an event log extraction is to learn how to recognize activity occurrences (events) in the SAP database; Buijs does not consider this and just lists how events can be retrieved. In general, the focus of my project is to look at the entire process of extracting an event log in SAP, *from extracting data, giving semantics to it and constructing the event log*.

In his application prototype, XES Mapper, the user can specify with SQL statements each *action*, i.e. attributes and properties that belong to a specific event. In SAP, events that accompany a certain activity are stored in the database and should therefore be retrievable in a similar way. Tailoring this idea further should ideally lead to a repository, as Buijs also mentions in his improvements, where for various processes it is known how to extract the event log. Furthermore, the case study he presented gives information about the different types of activities that are related to the *Purchase to Pay* process and how the activity occurrences can be retrieved from tables and/or fields. The change tables (CDHDR and CDPOS) are used for one activity (Change Order Line), but these, as well as the regular tables, could be more extensively used to allow for the identification of *more* different types of activities than is shown in the case study.

The XES Mapper prototype has been developed further by Buijs and included as *XESame* in the ProM 6 toolkit [23]. XESame allows a domain expert to extract the event log from the information system at hand without having to program.

3.4 Commercial Products

This section gives a short introduction to a couple of commercial products available. Some of these claim to be able to do process mining in SAP, some are just interesting because they provide support to create, identify and clarify the processes that can be implemented in SAP. A graphical overview of these *process mining tools* is given in Figure 3.2.

In the field of commercial process mining, Futura has few competitors. A tool that is build specifically for the extraction of event chains from an SAP database is the EVS ModelBuilder SAP Adapter, which is discussed in Section 3.4.1. Futura's main competitor is the ARIS toolkit from IDS Scheer. Although they do not offer real process mining techniques with



Figure 3.2: Process Mining Tools

their Process Performance Manager (Section 3.4.2), they have a broad range of software within the ARIS toolkit available which allows a company to gain insight in their processes. The ARIS Process Performance Manager tries to close the gap between business process design and SAP implementation. Another similar product is LiveModel, a product developed by Intellicorp, discussed in Section 3.4.3. More and more of these ‘tool vendors’ jump into the field of Business Process Management, but they all have their own challenges and are often complicated to use and understand; *user friendliness* is high on Futura’s list of priorities. Another company that is rapidly setting its name in the process mining world is Fluxicon, a company set up by two software engineers and PhDs in process mining. More information on them can be found in Section 3.4.4. A final section, Section 3.4.5, is dedicated to the SAP Solution Manager, which both the ARIS Process Performance Manager and Intellicorp LiveModel make use of.

3.4.1 EVS ModelBuilder

Started out as a research project by professors from the Norwegian University of Science and Technology, the Enterprise Validation Suite (EVS) is a visualization and process- and data mining framework [13], now commercially distributed by Businesscape. It allows for applying a combination of these techniques on event chains. Event chains are a more generic interpretation of traces, events in an event chain do not necessarily relate to a single process instance. For complex information systems like SAP it is easier to retrieve those event chains since there is not always a clear mapping between events and process instances. The EVS ModelBuilder allows a user to define a mapping on an SAP database in order to extract event chains. Process instances are constructed by tracing resource dependencies between executed transactions.

In [13] it is shown how the system is applied to extract and transform related SAP transaction data into an MXML event log. Van Giessel’s work builds on this principle, however, the complicating factor in using the EVS ModelBuilder remains the absence of a relation between events and a single process instance, each event needs to be defined explicitly. Furthermore, domain knowledge about each process is needed to be able to construct a correct mapping.

3.4.2 ARIS Process Performance Manager

The ARIS Process Performance Manager (PPM) is a product released by IDS Scheer. It is part of the ARIS platform and contributes to a solution for process-driven SAP management [12]. The advantage of the ARIS toolset is that it has a tight coupling with SAP. This means that SAP solutions are implemented using SAP reference processes available in the ARIS Business Architect for SAP. These implementations can then be synchronized with the SAP Solution Manager (Section 3.4.5). The PPM can visualize how processes are executed by using live data, and can reconstruct the execution of each business transaction from start to finish. The connection between the ARIS toolset and the SAP Solution Manager is done with the help of the SAP Java Connector. Communication to and from the SAP Java Connector to SAP is done by *Remote Function Calls* (RFC). RFCs form the standard SAP AG interface for communication between the SAP client and server over TCP/IP connections.

Details about the ARIS PPM are unfortunately difficult to obtain; it is not clear whether process mining is fully provided at the moment. In [14], a master study from 2006, a business process is analysed with three different software tools, including the ARIS PPM. It is shown that ARIS PPM does not support discovery as it is present in Reflect or ProM; it takes as input instance EPCs instead of event logs. Because of this, ARIS PPM depends on prior knowledge of the process, already incorporated in the EPC models. The emphasis in ARIS PPM is on performance calculation and KPI (Key Performance Indicator) reporting.

3.4.3 LiveModel

Similar to the ARIS toolset, Intellicorp's LiveModel¹ forms another environment for designing, evaluating and optimizing processes within a company. It uses the Visio Business Modeler to model SAP processes, and is integrated with the SAP Solution Manager to create the linkage between these business processes and SAP components. Like the Aris PPM, few detailed information is available about how the connection is made to the SAP Solution Manager, but we assume that this is also done by RFCs.

Like the PPM, LiveModel does not provide real process mining. The business processes are already available in some sort of environment, in this case the ARIS Business Architect or the Visio Business Modeler. Through a connection between these environments and the SAP Solution Manager, meaning is given to the different building blocks and related data can be retrieved from SAP. This provides the opportunity to map the data onto the process and simulate it.

3.4.4 Fluxicon

Fluxicon² is a small company set up by two PhDs from Eindhoven University of Technology, Dr. Anne Rozinat and Dr. Christian W. Günther, who have researched process mining and BPM for more than four years. The ProM toolkit is used for process mining, a product they both have worked on and still develop extensions for. Recently they developed a product of their own called Nitro. A tool for converting data in CSV and MS Excel files to event

¹<http://www.intellicorp.com/LiveModel.aspx>

²<http://fluxicon.com/>

logs, which in turn can be loaded into ProM. Furthermore, in collaboration with Eindhoven University of Technology they defined the new XES event log format [11].

While Futura is primarily focused around Futura Reflect, Fluxicon is engaged in a wider range of activities in the field of process mining and Business Process Management. A lot of consulting is done using ProM.

3.4.5 SAP Solution Manager

Another product from SAP AG is the SAP Solution Manager. It is a centralized solution management platform that provides the tools, the integrated content and the gateway to SAP that you need to implement, support, operate and monitor SAP Solutions [18]. It is a separate product that can be used in the early stages of a project. The business processes can be defined within the Solution Manager and coupled to and tested within SAP. Several business blueprints (i.e. process templates) are available to guide companies in designing their processes.

The Solution Manager is a nice tool to aid in designing processes, but cannot be used for this project. When analyzing data from a company, you cannot assume that the Solution Manager is used within the company. Besides that, the idea of process mining is to construct (discover) the process from data that is available, and not project the data on the process that is available (i.e. the solution manager does not discover a process, it executes data in a given process).

3.5 Concluding Remarks

This chapter has shown that there is a broad range of software available that gives companies insight in their SAP processes. Real Process Mining software for SAP is still not available and little research is done in this area. Van Giessel's work has the closest connection to my project but lacks several aspects and requires a lot of manual work. Buijs' work on extracting event logs from relational databases might help the most in this project, however, plenty of things could be tailored for SAP and added to the implementation. What distinguishes my project from previous research and software available is the following:

- The automatic proposal of a *case notion*. Since an SAP process more or less contains specific type of activities, the connection (if present) between these activity occurrences should be identified automatically (Chapter 6).
- Being able to *incrementally update* a previously extracted event log when new data is available (Chapter 7).
- A *repository* for SAP processes should be available which makes it easy to construct an event log for a specific process (Chapter 8).

The second bullet of the list above is an interesting one; very little research is done in updating event logs. This project makes use of some principles presented by Van Giessel and Buijs, but focuses on implementing and researching the above list. We furthermore try to use the power of the SAP system itself, i.e. learn to execute the SAP business processes ourselves and detect when and what changes have occurred in the underlying database.

Chapter 4

Extracting Data From SAP

This chapter describes two approaches that have been investigated during my project to retrieve data from SAP's database. Of course we could directly download the data from the underlying database, however, an alternative approach is considered in the light of supporting the *incremental updating of event logs*. This approach, described in Section 4.1, is a new idea and uses SAP Intermediate Documents to retrieve the data from the database. The second approach presented in Section 4.2 is more conventional and directly consults SAPs underlying relational database. Concluding remarks on these two approaches and how to continue from there is discussed in Section 4.3.

4.1 Intermediate Documents

SAP Intermediate Documents (**IDocs**) are standard data structures for Electronic Data Interchange (EDI) in SAP, between, for example, an SAP installation and an external application. They allow for asynchronous data transfer in SAP's Application Link Enabling (ALE) system.

4.1.1 Principle

Each IDoc that is generated consists of a self-contained text file that can be transmitted from SAP to the requesting workstation without connecting to the central SAP database. SAP offers a wide range of IDoc message types that can be configured. An example of such a message type is the IDoc *Orders*; this IDoc can contain information about purchase- or sales orders. With the help of these pre-defined message types, IDocs provide a clearly defined container to send and receive data. Each IDoc has a single control record; the structure of this record describes the content of the data records that will follow and provides administrative information (e.g. message type), as well as its origin (sender) and destination (receiver). IDocs can be generated at several points in a transaction process. When a user performs such a transaction, IDocs can be generated and passed to the ALE communication layer. This layer performs a Remote Function Call (RFC), using the port definition and RFC destination specified by the customer model.

Research was done on how the principle of IDocs can be used to construct an event log. The idea is to send IDocs, transparent to the user who executes the process, to an external logical system (e.g. my computer) whenever specific actions are done. Looking at the procurement

cycle, IDocs can be sent after creating a Purchase Requisition, creating a Purchase Order, changing a Purchase Order and much more. Having acquired all these IDocs on the external receiving system, the IDocs belonging to the same case identifier of the process should then be *tied* together to retrieve the concerning trace. In this way, the external system is continuously kept up to date about all actions that are performed within SAP.

4.1.2 Evaluation

To test this principle, a connection to an SAP installation is set up in a *logical system* at the receiver side with the SAP Java Connector (SAP JCo). A logical system is SAP terminology and is used to identify an individual client in a system, for ALE communication between SAP systems. The Java connector registers itself under a specific RFC destination to which messages can be send through EDI. The communication of messages is performed with the transactional RFC method (asynchronous communication), as depicted in Figure 4.1.

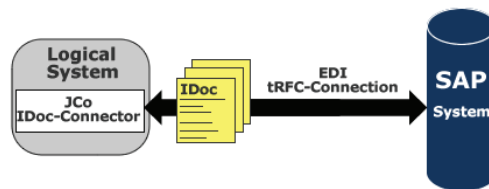


Figure 4.1: Principle of IDoc communication

The value of using IDocs to construct event logs, or other process analysis techniques, has not been investigated before and gives a new view on data extraction in SAP. This new approach appeared to be promising. The idea of using IDocs is to send messages after specific actions are done, and subsequently construct an event log upon receipt of all these messages. In the light of supporting incremental updating of events logs, the IDoc approach is very applicable. *Timestamps* of events play an important role in updating event logs; these inform us about the order of events. We could include a timestamp upon creation of each IDoc, this way the completion time of the activity is known. However, the following are the three most important issues encountered when trying to implement this approach:

1. IDocs can be configured in SAP to be sent after a specific action. By default often at most one outgoing communication method can be specified for each action (e.g. Fax, a Print Output, EDI). Thus, in real life situations, communication channels with vendors most probably need to be changed to be able to generate event logs, which is unacceptable.
2. The IDoc message types are specifically created for EDI communication, that is, they only contain information that is relevant for the receiver side, often a vendor. Creating the link between different IDocs that handle the same case is therefore not a trivial task, and even sometimes impossible due to missing information.
3. Setting up the IDoc approach will require extensive changes in an operational SAP installation.

All these drawbacks can be summarized as: *too much configuration is necessary* at the customer side to get this method to work. The IDoc method could work when customization

is allowed, something that plenty of companies do not allow due to license and warranty agreements of their SAP installation. Customization would allow for the sending of IDocs at any point in time. SAP provides the opportunity to debug, which enables a user to trace the exact line in the source code where a certain task is performed. The source code could be adapted in such a way that data is collected for the IDoc and send to a receiver at a specific point in the code/process. As for the second drawback mentioned, customization allows the user to create their own IDocs as well, such that the IDocs are filled with all data necessary to map the activity (specified in the IDoc) to a case identifier. All this however requires the user to be a SAP developer and make changes to the underlying SAP code.

These issues led to the fact that further research on IDocs was discontinued in this project. The solution would require too much configuration at the customer's side. Furthermore, the principle of IDocs would only be interesting when looking at performing incremental updates of event logs. Another approach (e.g. like in Section 4.2) should still be considered to *create the initial event log* with the historical data available.

4.2 Database Approach

Our approach in the previous section gathered data into an IDoc upon execution of a specific transaction. An alternative and frequently used method is to directly download the relevant data from SAP's underlying database. The relational database management system (RDBMS) in which this database resides can either be MaxDB or Oracle depending on the SAP installation. SAP MaxDB is the RDBMS developed and supported by SAP AG themselves, while Oracle is still the most widely used RDBMS within SAP. MaxDB is growing in popularity and focuses mainly on large SAP environments. With the help of transaction *DB02*, information can be retrieved about the database. In our IDES test system, Oracle is used as the RDBMS. A total of 73.407 tables are present that hold 87,9 gigabytes of data. The number of tables that is present differs from installation to installation, depending on the number of modules installed and the DB model view that is accessible.

4.2.1 Obtaining Data

To view the contents of a table in SAP, transaction *SE16* can be used. Upon specifying the table name, parameters can be set to narrow the search results. Figure 4.2 shows an excerpt of the EBAN table (Purchase Requisitions) that was retrieved by performing the SE16 transaction. Through *SE16* it is possible to download the table in various formats: Spreadsheet, Unconverted, Rich text format and HTML format. Upon selecting the download format, the table is created in this format and allocated in memory at the SAP server. It is important to download the data in the same format as that it resides in the SAP database; there exists some minor issues with specifying this download format, these can be found in Appendix B. After completion of the download, it can for example be loaded into a local database. A drawback of this approach is the *limited amount of memory* that is often available to prepare tables for download. Large tables should therefore be downloaded in separate parts. This issue stresses the need of having the possibility to incrementally update event logs; if we update an event log frequently we would not have these memory problems.

This downloaded data could also be acquired by directly connecting to SAP from an application. The Java Connector that is mentioned in Section 4.1.1 can execute specific commands

Data Browser: Table EBAN Select Entries 200

Table: EBAN
Displayed Fields: 10 of 138 Fixed Columns: 3 List Width 1023

	HANDT	BANFN	BNFPO	BSART	BSTYP	BSAKZ	LOEKZ	STATU	ESTKZ	FRGKZ	FRGZU	FRGST	EKGRP	ERNAM	ERDAT	TXZ01
☐	000	0010000003	00010	NB	B		X	N	R				001	HEREDITH	22.11.1996	Metallic Glaze
☐	000	0010000004	00010	NB	B		X	N	R				001	KEHNE	22.11.1996	Metallic Glaze
☐	000	0010000208	00010	NB	B		X	N	R			R1	002	KEHNE	22.11.1996	Services
☐	000	0010000384	00010	NB	B		X	B	F				007	STEIMER	08.02.1996	Durchfuehren von Reinigungsarbeiten
☐	000	0010000385	00010	NB	B			N	F				007	STEIMER	25.11.1996	Laufraud Typ 5A, elektrische Pumpe
☐	000	0010000418	00010	NB	B			N	F				007	STEIMER	25.11.1996	Modifiziertes Laufraud Typ II
☐	000	0010000563	00010	NB	B		X	N	F				007	NISTA	28.02.1995	Durchfuehren von Reinigungsarbeiten
☐	000	0010000707	00010	NB	B		X	N	R				001	SCHULTE	22.11.1996	XXXX
☐	000	0010000709	00010	NB	B		X	N	R				001	WAHL	22.11.1996	Twisted Pair Cable
☐	000	0010000709	00020	NB	B		X	N	R				001	WAHL	20.11.1998	Twisted Pair Cable
☐	000	0010000709	00030	NB	B		X	N	R				001	WAHL	20.11.1998	Twisted Pair Cable
☐	000	0010000744	00010	NB	B		X	N	R			R1	001	ISICC	22.11.1996	PC IBM-Standard

Figure 4.2: A screenshot from the EBAN table

to query the SAP database and download data. Visual Basic for Applications (VBA) in MS Excel also offers possibilities to connect to SAP. However, the same restrictions again apply: a limited amount of memory is available to prepare these tables for download. An interesting open source tool that deals with this problem is Talend¹. Talend's Open Studio Version 3.0 allows a user to create its own extraction process with pre-defined building blocks. These allow for example to connect to SAP and repeatedly extract data from specified tables.

As was mentioned in the IDoc approach, in the perspective of incremental updating of event logs, timestamps play an important role. When applying the database approach, we somehow have to be able to attach a timestamp to the data we download (e.g. that it contains data till timestamp t_1). This way, downloading new data (data till timestamp t_2) would concern data between two timestamps (t_1 and t_2). So it is important to retrieve the correct timestamp information from the SAP database (explained in detail in Chapter 7).

4.3 Conclusion

In this project we continue to acquire our data as explained in Section 4.2. This method enables us to download the data in a desired format and to put restrictions on the records to display and download. Furthermore, the downloaded files could be imported into a (Relational) Database Management System (DBMS) like MySQL or PostgreSQL in order to create a copy of the relevant part of the SAP database. This speeds up the process of querying the database and consulting data in the database.

The principle of using IDocs for data extraction is worthy to mention again. If full customization is allowed on the target SAP system, communication channels could be set up and configured between an extraction application and SAP, such that continuous event log extraction, and thus monitoring of processes, is possible. This however requires a very different approach than the one we consider in the rest of this project. Tailoring the IDocs approach could turn into a nice solution but requires more technical knowledge on SAP and available support within the SAP target system, something that is often not the case. An implementation of the IDoc approach would perfectly support the incremental updating of event logs.

¹<http://www.talend.com>

Chapter 5

Extracting an Event Log

Extracting an event log can be regarded as a *crucial step* in a process mining project. The structure and contents of an event log determines the view on the process and the process mining results that can be retrieved. In the previous chapters, the need for a generic event log extraction procedure for SAP processes was raised. In this chapter we present this procedure and delve deeper into important aspects that should be considered during event log extraction for an SAP process. It is important to be aware of the influence of decisions made in the event log extraction phase.

An important first step in the event log extraction procedure is to make some decisions about the process mining project at hand. This helps in mapping out the business process to be analyzed and avoids problems later on. Section 5.1 discusses this and presents the influences this step has on the structure of our event log. After this, we present our method for extracting an event log from SAP ECC 6.0. This method can be divided into smaller steps that together lead to an event log for a given SAP process. Section 5.2 gives a simplified graphical representation of this method. The accompanied subsections take a closer look at this procedure and explain the steps in detail. This starts with some preparation activities to collect information about a process; these should only been done *once for each business process* and can be found in Section 5.3. After that we outline how to process all this information and how to construct the event log from that point onward (Section 5.4). Do note that the *incremental updating* of event logs is *not yet considered* in this chapter. It is introduced as an extension of our normal extraction procedure in Chapter 7.

5.1 Project Decisions

Before we start an event log extraction we first need to determine the scope, goal and focus of the process mining project. This ensures that our event log contains the correct view on the process and we do not have to extract an event log repeatedly before the structure satisfies our expectations.

5.1.1 Determining Scope and Goal

The choice of the business process to extract implicitly determines where and what kind of information needs to be retrieved from the SAP system, i.e. it determines the **scope** of

the project. For example, the Order to Cash process focuses on *Sales Orders* and *Goods Movements*; in our SAP system the SD (Sales and Distribution) and WM (Warehouse Management) modules are therefore interesting, and MM (Materials Management) could possibly be left out of scope.

Accompanied with this, a **goal** should be set for the project. The output of a process mining phase can vary; several process mining techniques exist (see Section 2.2), each of which demands different information from the event log. The most common task in process mining, *process discovery*, would for example require few additional information (attributes) to be present in the event log, whereas an in-depth analysis of the process (e.g. *performance analysis*) requires a more extensive event log.

The *scope* of a process mining project is therefore specified by the targeted SAP *business process*. Additionally, the *attributes* contained in the event log lead to the fulfillment of the process mining project's *goal*.

5.1.2 Determining Focus

If a process is chosen, it might be interesting to **focus** on *specific parts* of that process *in detail*. In a corporate setting this would typically be done in agreement with a (Business) Process Manager or employee who actually execute the process. For example, it might be possible that a company detects several flaws around its shipment of goods activities. In this case it might be valuable for the company to add all activities related to shipments of goods to the process it wants to analyze. Using the CDHDR and CDPOS **change tables** in SAP, very detailed information can be acquired about when changes occurred, who was responsible and so on.

It is thus very important that the possibility exists to select activities in a process and to add new activities to that process in order to specify the level of detail. In the case studies presented in Chapter 9, all changes to Purchase Orders (excluding (un)deletion and (un)blocking of purchase orders) are for example captured in **one activity: Change Purchase Order**. This could easily be split up in several smaller activities like Changing the Order Quantity, Changing the Delivery Date, Changing the Supplying Vendor and Changing the Delivery Location.

5.2 Procedure

To create an event log for a given business process there are basically five important things we need to know: (1) the **activities** out of which the business process consists, (2) details on how to recognize an **occurrence** of such an activity, (3) the **attributes** to include per activity, (4) the **case** that determines the scope of the business process and (5) the **output format** of our resulting event log.

With an occurrence of an activity we indirectly mean an event. In process mining, an event specifies what activity occurred, when it occurred and by whom it is executed. The output format is more or less pre-defined by the process analysis tool that is used. Knowing how to recognize events and defining the event log format of the event log is something that

should be done in advance. Determination of the case and selection of activities is something that should be done during the actual performance of the event log extraction. Figure 5.1 presents a sequential flow diagram that outlines the basic procedure of extracting an event log for SAP.

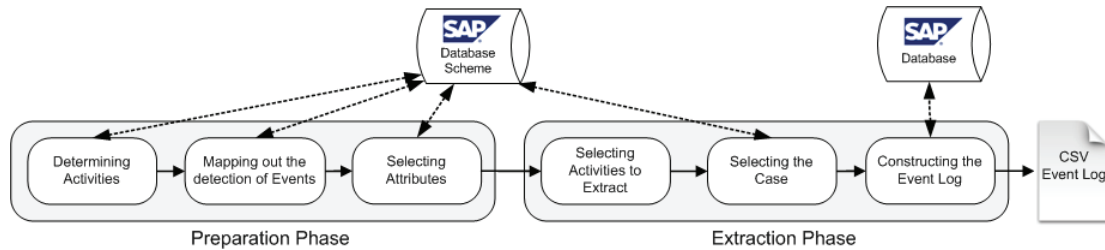


Figure 5.1: Basic Extraction Procedure

We split our procedure in a **preparation phase** (Section 5.3) that should be traversed **once** for each process, per type of project. This phase entails the collection of all SAP specific details. In the second phase, **extraction phase**, we actually obtain the event log. The obtaining of the log, explained in Section 5.4, can be done **repeatedly** with the information that is calculated during the preparation phase.

5.3 Preparation Phase

Each SAP process consists of several activities, Section 5.3.1 therefore presents the first step of the preparation phase, *determine activities*. In Section 5.3.2 we deal with how to *map out the detection of events* in SAP, that is, how can we observe in the SAP database that an activity has occurred. Section 5.3.3 discusses the *selection of attributes*; that is, the attributes which comprise our resulting event log.

5.3.1 Determining Activities

In order to mine a specific process in SAP, we need to select the set of relevant activities for this process. In Section 5.1.2 we stressed the importance of being able to select a subset of activities in a process, in this section we will go one step back and discuss how to determine all activities that should be selectable in such a set. We can thus select activities in two stages: **(1)** determining all activities that could exist in a process, and **(2)** in the extraction phase, be able to only look at a subset of this entire set of activities.

The table below sums up the primary sources of information that exist to determine this set of activities.

Table 5.1: Sources to Determine the Set of Activities

Standard	Corporate Environment
1. SAP Best Practices	4. Process Executor
2. SAP Easy Access Menu	5. SAP Consultant
3. Online Material	
6. Change Tables	

In our project, the four standard sources were consulted to get acquainted with SAP's *Purchase to Pay* and *Order to Cash* process. These sources can be considered generic enough to apply on other (standard) SAP processes. When performing an event log extraction in a corporate setting, additional sources might be consulted to become aware of the activities that are executed in the company's process.

Actually, our activity set determination consists of two or three stages. First, consulting information about the 'standard' SAP processes; second, in a 'corporate setting', discussing the process within the company, and third, tailoring this based on the scope, goal and focus of the project.

1. SAP Best Practices

The SAP Best Practices were already introduced in Section 2.1.3. Mainly used as reference models for the most common processes, they provide us with a detailed list of activities that occur in a process. Besides the PTP and OTC process, best practices exist for example for *Advanced Shipping Notification via EDI - Outbound*, *Non-Stock Order Processing*, *Purchase Rebate*, *Sales Returns* etc. A couple of best practices provide a (Microsoft Visio) flow diagram to gain more insight in the order of execution of activities within the process. Some processes include an additional document that lists the detailed steps that should be executed in SAP.

2. SAP Easy Access Menu

The home screen of SAP ECC 6.0, the Easy Access Menu, provides us with more information on a process than one might think. The Easy Access Menu is structured per module and thus holds transactions that are related to that module. Activities are performed by executing transactions and interesting activities should therefore be identified by its accompanying transaction. For example, activities in the PTP process are mainly performed through the Materials Management module (MM) and for the OTC process through the Sales and Distribution (SD) module. Common sense, experience, as well as the SAP best practices quickly guide you to which modules are involved in a process.

By expanding such a module, all accompanying transactions are listed and new interesting activities might thus be recognized. For example (see Figure 5.2), expanding the *MM module*, *Purchasing* and then *Purchase Order*, lists all transactions related to a Purchase Order. Due to the fact that the PTP process more or less centers around Purchase Orders, one can assume that all operations to a Purchase Order could be included in the PTP process. In the example this includes creating the Purchase Order (which can be done in various ways), releasing the Purchase Order, Changing the Purchase Order and other follow-up functions.

Not all 106.000 existing transactions can be found through the SAP Easy Access Menu, but for a simple user (and thus executor of a process) the most important ones can be found. Furthermore, not each transaction leads to an interesting activity. Transactions have an accompanied transaction code (see Section 2.1.2) to execute them, and which leads to a call to their related ABAP program. These programs could just be informative as well, like consulting a database (*SE16*) or checking the status of an IDoc (*WE02*).

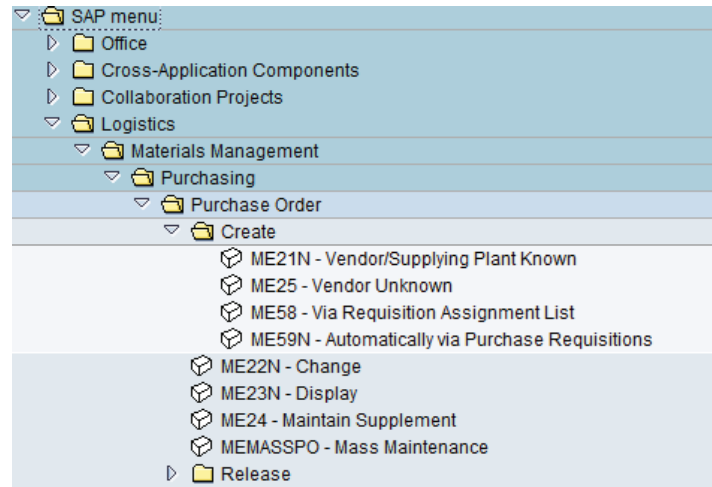


Figure 5.2: Excerpt from the SAP Easy Access Menu

3. Online Material

With large software packages like SAP ERP it is obvious that there are a large number of people using it, discussing it, researching it and in turn having problems with it. The Internet is an ideal location to post and discuss these, which makes it a very important source of information for SAP processes. By querying a process (e.g. Purchase to Pay), an abundance of information is found on this process, including its related activities. SAP itself has a large community network (SDN¹), which includes a forum to post and discuss problems, a wiki, eLearning options, Code Exchange and so on.

4. Process Executor

When handling real-life data (i.e. from a process executed within a real company), who other than the person executing the process in that company can give you more information? Together with that person you can discuss which steps of the process are performed and identify the important activities. A disadvantage of (only) consulting an in-house expert is that only the activities are identified that the expert is aware of. An interesting aspect of process mining is that *outliers* (special cases) can be detected, so you have to make sure that all relevant activities for the process are included, and traces that deviate from the standard process are detected as well.

5. SAP Consultant

The concept of an SAP consultant is well-known, in the first place because they are expensive to hire, but also because the tiniest change to an SAP installation might require an SAP consultant. SAP has a fixed structure that has been around for many years. The architecture behind SAP is still more or less as it was in the beginning years and the fast growth of SAP lead to the fact that the underlying architecture could not evolve with the exploding demand. Adaptations in the source code are difficult to make and often require an army of

¹<http://www.sdn.sap.com/irj/scn>

programmers. The good thing is that they are currently evolving to an E-SOA architecture (see Section 2.1), but the bad thing is that SAP is an ‘e-cement’, it is hard to get rid-off and you need to have a long term strategic view of the system.

SAP consultants are specialized in maintaining and/or implementing SAP software. They are experts in the field and often focus on one module. An MM SAP consultant for example has an enormous knowledge about the Purchase to Pay process and is easily able to tell you the various activities that exists in the process, what deviations exist and where to find them.

6. Change Tables

There are some other small tricks to get information about activities that exist within a process. Most of the time, consulting one (or more) of the five sources above is sufficient, but if you for example want to know everything about activities related to a Purchase Order, you can try another approach. Due to the fact that Purchase Orders are related to the EKPO and EKKO table, you could narrow down your search and look for changes on the EKPO and EKKO table in the change tables (CDHDR and CDPOS). Each change to these tables is probably related to a Purchase Order, so detailed changes to Purchase Orders could be tracked (like changing an order *delivery* date or changing an order *quantity*).

Result

The result of this Section (5.3.1) is the set of activities that occur in a given SAP process.

5.3.2 Mapping out the detection of Events

Knowing which activities are related to a process, what their base table is and how to execute them is one thing, but recognizing occurrences of these activities in the SAP database is a bit trickier. As mentioned earlier, with an occurrence of an activity we indirectly mean an event. In process mining, an event specifies what activity occurred, when it occurred and by whom it is executed. SAP stores an abundance of information in its database, but it is of vital importance to be able to give context to that data. This principle is nicely captured in the subtitle of a recent book on Business Intelligence [15], *Data is Silver, Information is Gold*. Finding your way in the SAP database is often a time-consuming task and interpreting the data requires a lot of knowledge about SAP. Very few information is available about the structure of the SAP database and how everything is related. Table and field names are often cryptic and difficult to understand which quickly makes you feel desperate.

In this section we present different ways to give meaning to SAP data (contained in the SAP database) by translating data to events (an activity has occurred). Like in Section 5.3.1, there are different approaches to do this. Most information is gathered by getting experienced with SAP and its processes, executing the related activities and checking whether, where and what changes occurred in the underlying database. In this project, the following methods were used in order of importance:

1. Literature Review
2. Monitoring the Change tables
3. Online information

4. Repository Information System (Table Relations)
5. Performing an SQL trace

1. Literature Review

By first analyzing other case studies or literature in this project we became familiar with event log extraction for SAP processes. In Buijs' and Van Giessel's work for example, a lot of information is available about the PTP process which helped us in identifying the occurrences of activities in SAP.

The mentioned relevant tables that are accompanied with an activity were analysed with transaction *SE16*. After performing an activity, we can browse through these tables, filter on a timestamp and check if records were added or updated. If this is indeed the case, we check what exactly is inserted into the table, how this can be distinguished from (possibly) other events that reside in the same table and how these events can thus be retrieved.

2. Monitoring the Change Tables

The change tables are a nice addition to the regular tables to detect events. To detect whether an activity leads to a change (event) in the change tables you can simply execute the activity (by performing the corresponding transaction) and afterwards consult the *change header table* (**CDHDR**) with transaction *SE16* to check whether the activity has occurred on the given timestamp. If it has occurred you can take note of the **changenr** that is accompanied with the event and look up this number in the *item table for change documents* (**CDPOS**). CD-POS gives you insight in what values exactly have been changed by performing the activity, while the header gives you some more general information for the change. Information from both these tables allows you to recognize the occurrence of certain activities (events).

Figures 5.3 and 5.4 present some more insight in this idea. From the CDHDR table we retrieved all records that occurred on date **28.10.2010** between time **15:00:00** and **17:00:00**, and can observe that user **IDADMIN** executed transaction **ME22N** (*Change Purchase Order*) on **15:26:31**. The change number that is related to this event is **0000591522**.

MANDANT	OBJECTCLAS	OBJECTID	CHANGENR	USERNAME	UDATE	UTIME	TCODE	PLANCHNGNR	ACT_CHNGNO	WAS_PLANND	CHANGE_IND	LANGU	VERSION
800	BANF	0010013596	0000591518	IDADMIN	28.10.2010	15:03:38	ME57				U	E	000
800	BELEG	800100051000000052006	0000591528	IDADMIN	28.10.2010	16:48:49	MRBR				U	E	000
800	BELEG	800100051000013722003	0000591530	IDADMIN	28.10.2010	16:49:26	MRBR				U	E	000
800	EINKBELEG	4500017217	0000591517	IDADMIN	28.10.2010	15:03:25	ME21N				I	E	000
800	EINKBELEG	4500017217	0000591519	IDADMIN	28.10.2010	15:14:41	ME22N				U	E	000
800	EINKBELEG	4500017218	0000591520	IDADMIN	28.10.2010	15:18:38	ME21N				I	E	000
800	EINKBELEG	4500017218	0000591521	IDADMIN	28.10.2010	15:23:07	ME22N				U	E	000
800	EINKBELEG	4500017218	0000591522	IDADMIN	28.10.2010	15:26:31	ME22N				U	E	000
800	EINKBELEG	4500017218	0000591523	IDADMIN	28.10.2010	15:28:58	ME22N				U	E	000
800	EINKBELEG	4500017218	0000591524	IDADMIN	28.10.2010	15:32:38	ME22N				U	E	000
800	EINKBELEG	4500017219	0000591525	IDADMIN	28.10.2010	15:42:20	ME21N				I	E	000
800	INCOMINGINVOICE	51050068712003	0000591529	IDADMIN	28.10.2010	16:49:26	MRBR				U	E	000
800	INCOMINGINVOICE	510500687222006	0000591527	IDADMIN	28.10.2010	16:48:49	MRBR				U	E	000
800	INFOSATZ	5300002591	0000591516	IDADMIN	28.10.2010	15:03:27	ME21N				U	E	000
800	LIEFERUNG	0180000111	0000591526	IDADMIN	28.10.2010	16:23:03	VL32N				U	E	000

Figure 5.3: Excerpt from the CDHDR table

The next step is to look up this change number in the CDPOS table. If we use transaction *SE16* and filter on change number 0000591522, two records are returned. This means that, due to the execution of this transaction **ME22N**, two things have changed. The first change is in table *EKPO*, the value of field *LOEKZ* changed from (L) to (). The *TABKEY* field

points us to the involved purchase order in table EKPO. The second change also occurs in EKPO, the field *STAPO* changed from (X) to (). Both LOEKZ (deletion indicator) and STAPO (statistical indicator) are thus changed. The LOEKZ field in EKPO has a value of ‘L’ when the corresponding order (line) is deleted. From the records in Figure 5.4 we can therefore conclude that an **Undeletion of a Purchase Order** has taken place on 28.10.2010 at 15:26:31 by user IDADMIN. A change of the statistical indicator alone does not give us information whether an undeletion has taken place, while the deletion indicator does.

	MANDANT	OBJECTCLAS	OBJECTID	CHANGENR	TABNAME	TABKEY	FNAME	CHNGIND	TEXT_CASE	UNIT_OLD	UNIT_NEW	CUKY_OLD	CUKY_NEW	VALUE_NEW	VALUE_OLD
☐	800	EINKBELEG	4500017218	0000591522	EKPO	800450001721800010	LOEKZ	U	1						L
☐	800	EINKBELEG	4500017218	0000591522	EKPO	800450001721800010	STAPO	U	1						X

Figure 5.4: Excerpt from the CDPOS table

Caution must thus be taken when analyzing the Change tables. Activities may lead to various changes in the change table and sometimes the same type of change may refer to different activities. It is therefore important that when retrieving activity occurrences from the change tables, you ensure that only one type of activity is retrieved.

On the contrary, another scenario that may occur is that after performing an activity, changes to the change tables have taken place, but it is impossible to relate these changes to a certain type of activity because essential information is missing. This is again due to the fact that not all changes are logged by default in the change tables. Performing an activity might lead to changes in the change table, but the essential information (that enables us for example to link the change to a specific Purchase Order or Invoice) might be missing.

Please note that it is possible that an activity can be detected by looking at the change tables *as well as* the regular tables. In this case, the option that provides the best performance should be chosen. Furthermore, not all activities can be detected from the change tables, depending on the SAP installation and configuration, system managers may chose to track all changes or even nothing. However, the standard configuration keeps track of the most important changes and is almost always implemented.

3. Online Information

Simply querying the SAP activity for which you want more information on the Internet quickly gives you more information than one might wish. With thousands of users and people customizing and configuring SAP, discussions can be found on various processes and activities, which often state references to the table and/or information we are looking for.

4. Repository Information System (Table Relations)

SAP’s own Repository Information System (RIS, accessible through transaction *SE84*), might also be of help. We specifically focus on the foreign keys we can retrieve for a table. Let us take the case where you for example do not know where a purchase requisition is stored, but you do know where a purchase order is stored. Suppose there is a reference to a purchase requisition in that record of the purchase order, you can then try to find the relation between

the column that holds this *purchase requisition reference number* and another table (= the table we are looking for).

5. Performing an SQL Trace

The last resort, if the methods above showed no results, is to turn on an **SQL trace** in SAP. This can be done by accessing *System* → *Utilities* → *Performance Trace*, checking *SQL Trace* and clicking *Activate Trace*. From that point onward, a log is maintained that holds all SQL queries that are performed by the SAP system. And with all, we mean all, that is each request SAP makes to its database is logged. It is therefore recommended to only switch on the SQL trace just before the end of performing an activity (often pushing the *Save* button), and then deactivating it after the save action. In the same menu where you activated and deactivated the SQL trace, you can chose *Display Trace*; this shows a list of all queries that are performed during the ‘Save’ action. This is still quite a lot since ‘side-actions’ are logged as well. By browsing through this list you can find out in which table(s) (relevant) records are inserted. A method to do this is to only look at SQL INSERT statements, and check if the INSERT values match what was filled in when performing the activity. If you then find the involved table, the next step is to look at the various records of that table and analyze how the occurrence of such an activity can be retrieved.

Future research could possibly investigate this approach further. More specifically: how can you automatically derive an SQL query, from a list of SQL queries that was retrieved by performing an SQL trace, that retrieves occurrences of the activity traced. A precondition for this is that all SQL statements in that list were logged as a result of executing one activity (i.e. there exists no ‘noise’ from other users/activities).

Result

The result of this Section (5.3.2) is for each activity a method to retrieve a list of occurrences for that activity.

5.3.3 Selecting Attributes

Events in an event log typically contain information about the **case identifier**, **activity name**, **executor** and **timestamp** of the event. This information is sufficient to construct a process model. However, when analyzing the process it is useful to have additional information about an event immediately available in the log, instead of having to look it up elsewhere. Futura’s CSV event log format (Section 8.1.2) allows for the addition of attributes, on the case and the event level.

As mentioned in Section 5.1.1, different goals may require different attributes. Consider a process where flaws are suspected in financial transactions. For each event, it then is important to include attributes related to payments and/or the amount of money that is attached to the case. Futura Reflect gives much attention to this. An extensive framework is developed to set filters on attributes and/or activities to analyze cases or events in detail. Our prototype should therefore have the possibility to define the attributes that need to be extracted per activity such that these can be included in the event log.

Result

The result of this Section (5.3.3) is the set of attributes that should be included in the event log.

5.4 Extraction Phase

The extraction of the log is performed after the preparation phase. Now that we have determined the outline of our process and collected all information, we have the possibility to extract an event log. This can be done **repeatedly** and starts with *selecting activities to extract* (Section 5.4.1), to specify the activities that should be considered within the process. This is followed by *selecting the case* to determine the view on the business process (Section 5.4.2). If the case is known, we set up a connection with the SAP database and start *constructing the event log* in Futura's CSV event log format (Section 5.4.3).

5.4.1 Selecting Activities to Extract

In the preparation phase we outlined how to determine the set of relevant activities for an SAP business process (Section 5.3.1). In the extraction phase we can narrow this set and only select the activities we want to consider in our event log extraction. This second time of 'selecting activities' is there ensure the desired view on the process is obtained and the focus is correctly set.

Result

The result of this Section (5.4.1) is a subset from all activities in the selected SAP process.

5.4.2 Selecting the Case

With traditional process mining techniques, an event log contains only one type of case that identifies to which process instance events belong. This case has to be determined and is often indirectly inferred from the scope and focus that were set for the project. In SAP, thousands of processes exist, which makes the selection of a correct case very difficult. For the most common processes, like the Purchase to Pay and Order to Cash process, the cases are often obvious and few candidates exist. When choosing the Purchasing Document as the case throughout the PTP process, all activities are extracted from a *purchasing document point of view*, whereas more detailed information could be gained when analyzing from a *purchase order line item point of view*. Other possible cases in SAP are for example a *sales order*, a *sales inquiry* or a *goods receipt*.

When only looking at activities that are directly related to one case, it is easy to determine the case. When more complex and larger processes are analyzed, which handle several types of documents and business objects, determining a case is a bit trickier and more candidate cases exist. The biggest challenge in extracting an event log for an SAP process is therefore to determine a valid case that is related to all activities.

Chapter 6 is completely devoted to the selection of a case and the influences this has on the view on the business process. It presents a procedure to automatically propose a case

for the business process by using the relations that exists between tables in the SAP database.

Result

The result of this Section (5.4.2) is a user selected case. Each event in the event log will be an instance of this case.

5.4.3 Constructing the Event log

The second step in the extraction phase, the final step in our event log extraction procedure presented in Section 5.2, is to construct the event log by querying the SAP database. This is based on the results from the previous sections. The event log can be extracted using the following (simplified) procedure for a given set of activities A (as calculated in Section 5.4.1).

- | | |
|----------------------|--|
| Section 5.4.2 | 1. Select a case for A |
| | 2. For each activity $a \in A$ |
| Section 5.3.2 | 3. Retrieve occurrences of activity a and store results in R |
| | 4. For each record $r \in R$ |
| Section 5.3.3 | 5. Extract relevant attributes att from r |
| | 6. Write att to an event log |

If a line (step) in the procedure above is supported by one of the previously presented sections, a reference to that section is given besides that line. In Chapter 8, a prototype is presented that implements this entire procedure. In that chapter we also delve deeper into the technical implementation and explain how the information from the preparation phase can exactly be translated to a querying language in order to construct an event log.

Furthermore we have to assume that only activity occurrences can be extracted that result in a change in the database. This is also one of the preconditions to apply process mining: execution of activities should be logged by the system.

5.5 Conclusion

Chapter 5 presented a key part of this project: the method for extracting an event log from SAP ECC 6.0. Roughly we can describe the method as follows: (1) a process is chosen and all activities for that process are determined, (2) activity occurrences in SAP are detected and can be retrieved, (3) the attributes that comprise the event log are specified, (4) the relevant activities to consider are selected, (5) the case to be used is determined and (6) the event log is constructed and stored in CSV format.

Our approach could be improved by considering the *automated discovery of events* by checking for patterns, focussing on timestamps, in the SAP database. There are thousands of timestamps in the SAP database; an approach could be developed that does not know what activities exists in a process, but discovers, interprets and extracts occurrences of new activities. Another similar method entails the performing of an SQL trace during execution of an activity; in depth analysis of the sequence of SQL statements performed could provide knowledge in how to detect activity occurrences.

Chapter 6

Case Determination

As mentioned in Section 2.2, event logs are structured around cases. The chosen case indirectly defines the way we look at the process. Each instance of the case uniquely identifies cases that flow through the process. Workflow Management Systems are typically build around the concept of cases, but processes in SAP do not have a pre-defined case. An important step in extracting an event log for a specific SAP process is therefore to determine the case that is used in the event log.

In the procurement process we introduced in Section 2.1.3, a case would typically correspond to a *purchase order*. However, the procurement process can also be analysed on a lower level, that is for *purchase order line items*. For the entire procurement process there are a few case notions that can be used throughout the entire process (like purchase order and purchase order line). Generally we can define the applicability of a case as follows:

A case is a valid case for an event log if there is a way to link each event in the event log to exactly one instance of that case.

When looking at specific parts (subprocesses) of the procurement process, many more notions of a case could exist (e.g. purchase requisition or payment). These additional cases can not be used for the entire process because we are unable to link all activities to such cases. For example, a payment is related to an order, and not to a purchase requisition. It is very important to be able to distinguish and detect these different case notions to allow the process to be examined on different levels. When a (part of a) process is unknown or new, it is often difficult to determine a case notion. Furthermore, if multiple case notions exist for a process, people are often unaware of this. This makes it necessary to support the (automated) discovery of case notions.

In this chapter we present a method to propose possible cases for a given set of activities (Section 6.1). These candidates are referred to as table-case mappings and are computed automatically. A common problem with SAP ERP (or other data centric ERP systems) is the issue of events not referring to a single process instance. The influence the case has on this issue is extensively discussed in Section 6.2. Ongoing research, presented in Section 6.3, is investigating new approaches to tackle this problem. We conclude in Section 6.4 by recapitulating everything and evaluating our table-case mapping approach.

6.1 Table-Case Mapping

This section describes a method to automatically retrieve the possible cases for a given set of activities. The meaning of the case (e.g. that it represents a purchase order) is often the same for each activity throughout the process, but for each table involved we may have a different way of identifying the case. The way we represent our case is therefore a bit more complex and is represented by a **Table-Case Mapping**. For each table, the Table-Case mapping provides fields in the table that (together) identify the case. The construction of this Table-Case mapping is built on the principle of table relations and foreign keys and is explained and presented step by step in the sections below.

6.1.1 Base Tables

A first step in determining the relations between activities is to identify the **base tables** in which information about the activities is stored. The base table for an activity is the table where the most important information for that activity is stored. For example, creating a *Purchase Requisition* produces a new record in the *EBAN* table. The base table we identify for the activity Create Purchase Requisition is thus EBAN. In Section 5.3.2, more information can be found on how the required information for activities can be retrieved in SAP, like what the base table is for an activity. Table 6.1 gives a mapping from some activities from the procurement process to their base tables.

Table 6.1: Activity to Table mapping

Activity	Table
Create Purchase Requisition	EBAN
Change Purchase Requisition	EBAN
Delete Purchase Requisition	EBAN
Undelete Purchase Requisition	EBAN
Create Request for Quotation	EKPO
Delete Request for Quotation	EKPO
Create Purchase Order	EKPO
Block Purchase Order	EKPO
Unblock Purchase Order	EKPO
Goods Receipt	MSEG
Invoice Receipt	RSEG
Payment	BSEG
...	...

We observe that activities that handle the same *object* have the same base table. For example, all activities related to *Purchase Requisitions* have as base table *EBAN*. Occurrences of activities can be detected in different ways, and also sometimes from different tables. The base table that you associate with an activity should therefore be the table from which you retrieve the activity information.

Base tables often have **header tables**; a header table contains a primary key that is referenced by at least one foreign key in the base table. This relationship between tables enforces referential integrity among the tables. Header tables are needed because they contain information like the timestamp and executor of (a couple of) events in the base table; these

header tables can be ‘discovered’ by following the foreign keys in the base table. For the tables in Table 6.1 we can for example identify the following header tables:

Table 6.2: Base Tables and their Header Table

Base Table	Header Table
EKPO	EKKO
MSEG	MKPF
RSEG	RBKP
BSEG	BKPF

6.1.2 Foreign Key Relations

The next step in finding the common case between activities is to identify the *relations* that each of these base tables have with other tables. Unfortunately, retrieving these relations must be done by hand since SAP does not present an easy interface for that. Relations between tables can be retrieved in the form of **foreign keys** and can be consulted with the **Object Navigator** through transaction *SE84*. A kind of Entity-Relationship Diagram (ERD) for a specific table can be retrieved from the ABAP dictionary (ABAP Dictionary → Database Tables → Graphic → Environment → Data Browser). Figure 6.1 presents this ERD for the table EKET (Scheduling Agreement Schedule Lines).

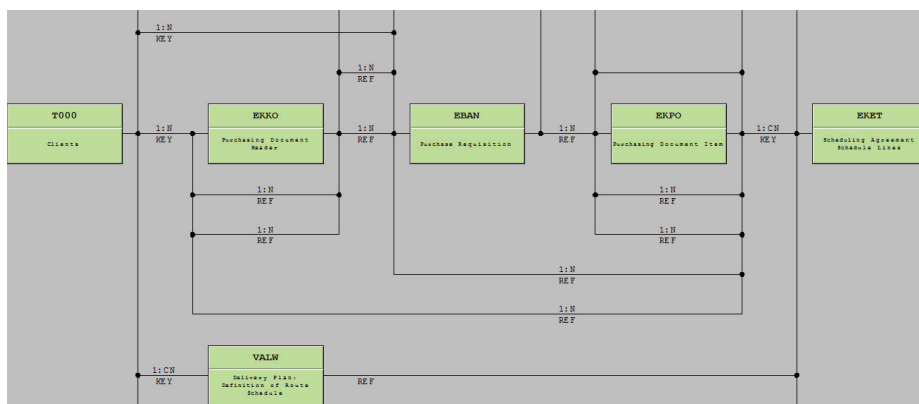


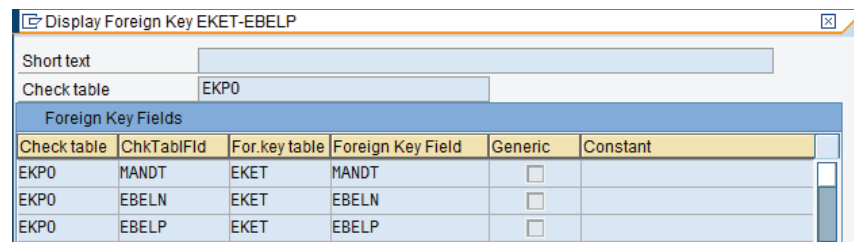
Figure 6.1: Relations EKET table

This diagram shows the relations from table EKET to other tables. If there exist relations in between those ‘other tables’ they are automatically included as well. Relations are represented by lines; the cardinality of the relation is included for each line. For example, there is a relation between table EKET and EKPO with cardinality **1:CN**. This means that in this relation an entry from table EKPO **must exist** for each entry in EKET (i.e. 1), and each record in EKPO has **any number of dependent records** in EKET (i.e. CN): this symbolizes a *one-to-many* relation. The cardinality **1:N** can be found in the diagram as well, the difference with 1:CN is that here **at least one dependent record** must exist.

In the diagram the relationships (lines) are bundled, this means that lines may overlap and it might not always be clear which tables are linked. Bundling of relations can be set on or off to cope with this problem. The relations present themselves in the form of **foreign**

keys. Details about a specific relation can be retrieved by double clicking the connecting line in the diagram, this shows the foreign key that is involved in this relation. For tables with many connections to other tables (many foreign keys) this is a time consuming task, but luckily this has to be done only once for each table. Tables can also have a foreign key with themselves, this happens when some fields (not the primary key fields) in a record of a table are linked to the primary key fields of a record of that same table. In Figure 6.1 we can observe for example that there exists three reflexive relations for table EKPO (two below and one above the table entity).

Continuing with our example from the EKET table, the foreign key that exists between the EKET and EKPO table is presented in SAP as follows:



Check table	ChkTabFld	For.key table	Foreign Key Field	Generic	Constant
EKPO	MANDT	EKET	MANDT	☐	
EKPO	EBELN	EKET	EBELN	☐	
EKPO	EBELP	EKET	EBELP	☐	

Figure 6.2: Foreign Key EKPO - EKET

The **foreign key table** is **EKET** and our **check table** is **EKPO**, this means that one record of the EKPO table uniquely identifies one record of the EKET table. The fields **MANDT**, **EBELN** and **EBELP** are related to the **primary key fields** of table EKPO, which in this case happens to have the same field names (MANDT, EBELN, EBELP).

Furthermore, in this case the fields of the foreign key table form the primary key for the foreign key table as well. This is not always the case; Table 6.3 presents a simple example of a foreign key relation between EKPO (Purchasing Document Item) and MARA (Material Master: General Data). The primary key of EKPO consists of MANDT, EBELN and EBELP, so **not** MANDT (Client) and EMATN (Material Number). The field names of the check- and foreign key table differ as well in this case, the primary key of MARA consists of MANDT and MATNR, while MATNR (material number) is represented by EMATN in EKPO.

Table 6.3: Example of a Foreign Key Relation between MARA and EKPO

Check table	Check Table Field	Foreign Key Table	Foreign Key Field
MARA	MANDT	EKPO	MANDT
MARA	MATNR	EKPO	EMATN

Now that we know how to extract foreign key relations from SAP, we retrieve all the foreign key relations for the base tables we identified. Besides these base tables, we extract the foreign key relations for related tables as well. With related tables we mean *header tables* or other lookup tables. For example, **BKPF** is the Accounting Document Header table (*related table*), whereas **BSEG** is the Accounting Document Segment table (*base table*). These header tables are often consulted to retrieve additional information about a record in the base table (required for our event log), thus the link between header- and base table needs to be known.

6.1.3 Computing Table-Case Mappings

The last section showed us how to retrieve the foreign key relations for all tables. For the tables in the procurement process this gives us about 620 unique relations. These foreign key relations are stored together for *all tables* such that it is possible to extract all candidate cases for a subset of these tables as well.

Let FK be the set in which all our foreign keys are stored; we can compute the Table-Case Mappings (returned in *Result*) for a given set of tables T by performing the algorithm `ComputeTableCaseMappings` with parameter T .

`ComputeTableCaseMappings( $T$ )`

1. $Result := \emptyset$
2. $Keys := \emptyset$
3. **for each** pair of tables (T_1, T_2) in the set T , $T_1 \neq T_2$
4. get each foreign key relation between (T_1, T_2) from FK and add to set $Keys$
5. **for each** $f \in Keys$
6. $\varphi := f$
7. $Result := Result \cup \text{TableCaseMapping}(\varphi)$
8. **return** $Result$

`TableCaseMapping( $\varphi$ )`

1. **if** φ covers all tables in T **then**
2. **return** φ
3. **else**
4. $R := \emptyset$
5. **for each** $g \in Keys$
6. **if** g and φ can be **merged**
7. $R := R \cup \text{TableCaseMapping}(\text{merge}(g, \varphi))$
8. **return** R

The algorithm `ComputeTableCaseMappings` computes all possible table-case mappings; it is supported by algorithm `TableCaseMapping`. For example, `TableCaseMapping( $f$ )` computes all table-case mappings that can be retrieved by starting with foreign key f . The result of the two algorithms above can be captured in the following definition:

$$Result = \bigcup_{f \in Keys} \{\text{TableCaseMapping}(f)\}$$

The first four lines of the algorithm `ComputeTableCaseMappings` create a set $Keys$ with all foreign key relations for the given set of tables T . This is done from the foreign key relations that are extracted in Section 6.1.2. The following paragraphs explain the two algorithms in detail, especially the concepts of **merging**.

Line 6 of the algorithm `ComputeTableCaseMappings` introduces the set φ . The elements in this set map tables to a list of fields within that table and is formally defined as follows:

$$\varphi :: \{T_i \rightarrow (F_1^i \dots F_n^i)\}, \text{ with } \varphi_i = T_i \rightarrow (F_1^i \dots F_n^i)$$

φ is used in both algorithms, below we explain three involved lines in detail:

ComputeTableCaseMappings

(line 6) Suppose $f = T_1(F_1^1 \dots F_n^1) \rightarrow T_2(F_1^2 \dots F_n^2)$
 $\Rightarrow \varphi := f \quad \equiv \quad \varphi := \{T_1 \rightarrow (F_1^1 \dots F_n^1), T_2 \rightarrow (F_1^2 \dots F_n^2)\}$

TableCaseMapping

(line 6) Suppose $g = A(X_1 \dots X_n) \rightarrow B(Y_1 \dots Y_n)$, then, g and φ can be **merged** iff:

$$\begin{aligned} & (1) (\forall i : 1 \leq i \leq |\varphi| : B \neq T_i) \wedge (\exists i : 1 \leq i \leq |\varphi| : T_i = A \wedge F_1^i = X_1 \wedge \dots \wedge F_n^i = X_n) \\ \vee & (2) (\forall i : 1 \leq i \leq |\varphi| : A \neq T_i) \wedge (\exists i : 1 \leq i \leq |\varphi| : T_i = B \wedge F_1^i = Y_1 \wedge \dots \wedge F_n^i = Y_n) \end{aligned}$$

(line 7: **merge**(g, φ)) **if** (1) is true: $\varphi := \varphi \cup \{B \rightarrow (Y_1 \dots Y_n)\}$
if (2) is true: $\varphi := \varphi \cup \{A \rightarrow (X_1 \dots X_n)\}$

Although foreign keys can be self referential (referring to the same table), with line three we ensure that these are not considered. These self referential keys are of no added value for the processes we analyzed (PTP, OTC). The definition of the merge maintains this idea, it ensures that φ only contains one entry for each table.

The resulting set *Result* contains all table-case mappings (i.e. φ 's) that are calculated. These were computed by looping over each foreign key, and recursively trying to merge this foreign key with other foreign keys. Let l be the size of the set *Result*, *Result* has the following property:

$$Result :: \{\varphi_i \mid 0 \leq i \leq l \wedge \neg(\exists j : 0 \leq j \leq l : j \neq i \wedge \varphi_i = \varphi_j)\}$$

Where:

$$\begin{aligned} \varphi_i = \varphi_j \Leftrightarrow & \quad \forall (S_x \rightarrow (X_1^x \dots X_n^x)) \in \varphi_i : \\ & (\exists (T_y \rightarrow (Y_1^y \dots Y_n^y)) \in \varphi_j : i \neq j \wedge S_x = T_y \wedge X_1^x = Y_1^y \wedge \dots \wedge X_n^x = Y_n^y) \end{aligned}$$

The more tables that are contained in our starting set T , the fewer table-case mappings are returned since the (common) connection between these tables is more difficult to make. An example of **one merge** can be found in Figure 6.3. Here, f (a foreign key between EKPO and EBAN) and g (a foreign key between EKPO and LIPS) are merged to φ (connecting EKPO, EBAN and LIPS). In subsequent merges f would be replaced with φ , and φ possibly extended with a new g .

Summarizing all of the above, we try to *connect* as much tables as possible through their foreign keys. The merged keys we retrieve is what we call **Table-Case Mappings**. Such a case identifier in the table-case mapping is for example composed of three fields (*Client*, *Purchasing Document Number* and *Purchase Order Line Item*), where each of these fields can thus be represented by an (other) column for each table. For example, Purchase Order Line Item is EBELP in EKPO, while it is identified by LPONR in EKKO. Table 6.4 presents three out of eight table-case mappings that can be retrieved for the chain of activities: Cre-

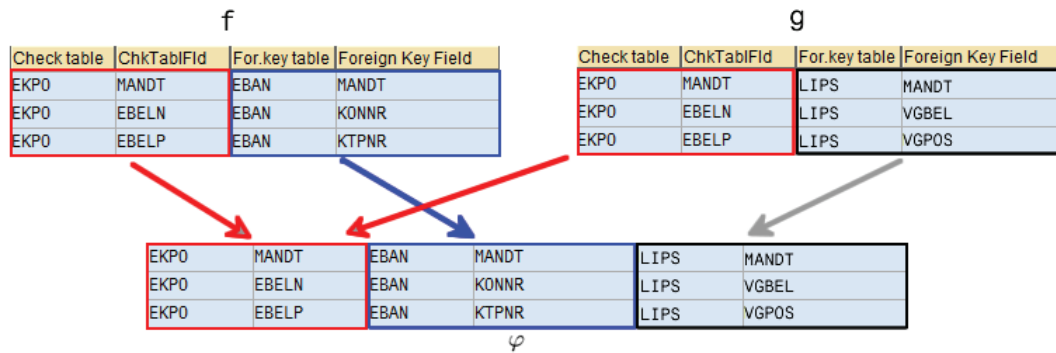


Figure 6.3: Merging two Foreign Keys

ate Purchase Requisition, Create Purchase Order, Create Shipping Notification, Issue Goods, Goods Receipt, Invoice Receipt and Payment to Vendor. Each table-case mapping in this table represents a notion of a case. In each line of a mapping, the columns that identify a key are separated by hyphens. In the first table-case mapping we see for example the lines **LIPS:** (MANDT - VBEL - VGPOS) and **MSEG:** (MANDT - EBELN - EBELP), this means that a combination of (MANDT, VBEL, VGPOS) values for a record from LIPS refers to the same object in MSEG that has those same values in their (MANDT, EBELN, EBELP) fields.

Table 6.4: Example of Table-Case Mappings

Table-Case Mapping 1	EKPO: (MANDT - EBELN - EBELP) EKBE: (MANDT - EBELN - EBELP) LIPS: (MANDT - VBEL - VGPOS) MSEG: (MANDT - EBELN - EBELP) BSEG: (MANDT - EBELN - EBELP) RSEG: (MANDT - EBELN - EBELP) EBAN: (MANDT - EBELN - EBELP) EKKO: (MANDT - EBELN - LPONR)
Table-Case Mapping 2	EBAN: (MANDT - KONNR - KTPNR) EKPO: (MANDT - EBELN - EBELP) EKBE: (MANDT - EBELN - EBELP) LIPS: (MANDT - VBEL - VGPOS) MSEG: (MANDT - EBELN - EBELP) BSEG: (MANDT - EBELN - EBELP) RSEG: (MANDT - EBELN - EBELP) EKKO: (MANDT - EBELN - LPONR)
Table-Case Mapping 3	BSEG: (MANDT - EBELN) EKKO: (MANDT - EBELN) LIPS: (MANDT - VBEL) EBAN: (MANDT - EBELN) MSEG: (MANDT - EBELN) RSEG: (MANDT - EBELN) EKPO: (MANDT - EBELN) EKBE: (MANDT - EBELN)

Interpreting Table-Case Mappings

The table-case mappings that are returned are a combination of check table fields and foreign key table fields. Take note that different cardinalities exist within foreign keys. For example, in EKKO there is only one unique record with the value ($MANDT = x, EBELN = y, LPONR = z$), whereas in BSEG multiple records could exist with that same combination of values ($MANDT = x, EBELN = y, EBELP = z$). Furthermore, the fact that we are merging multiple foreign keys, each having different cardinalities, magnifies this issue. This concept, known as **divergence**, including the consequences it has, is discussed in detail in Section 6.2 together with a similar issue: **convergence**.

It is possible to have NULL-values when looking at the actual field values in a table-case mapping. We just have to ignore these values and not consider the activities that are determined from the concerned table. In a process model this would be visible by a trace that does not contain activities that should be retrieved from that table. The fields in a table-case mapping therefore just represent how we can identify each case instance in a table, but does not guarantee that each case instance exists within a table.

Continuing with Table 6.4, we can see that a total of eight tables are present in each table-case mapping. The case identifier in **table-case mapping 1** consist of three attributes: **Client**, **Purchasing Document Number** and **Purchase Order Line Item**, where the fieldname for each attribute varies per table. In **table-case mapping 2** the same references to attributes are found (i.e. a Client, Purchasing Document Number and a Purchase Order Line Item), but their meaning is slightly different. The difference is with the attributes identified for EBAN. Table 6.5 lists the meaning of these attributes. In table-case mapping **1**, records from (EBAN) are selected where a purchase requisition is linked to a purchase order, whereas when table-case mapping **2** is chosen, records are selected where the purchase requisition is linked to a purchase order that is an outline agreement (e.g. a contract with a vendor for a predetermined order quantity or price). The table-case mapping approach thus ensures us that only one context (one table-case mapping) in which we look at the case is chosen.

Table 6.5: Attribute Values EBAN

Table	Field	Description
EBAN	MANDT	Client
EBAN	EBELN	Purchase Order
EBAN	EBELP	Purchase Order Item
EBAN	KONNR	Outline Agreement
EBAN	KTPNR	Principal Agreement Item

Table-case mapping 3 presents us another view on the process, here we choose the **Client** and **Purchasing Document Number** as the case identifier. If we choose mapping **1** or **2** as the case identifier to be used, we examine the process on a **purchase order line level**, whereas choosing mapping **3** leads to an analysis on a **purchasing document level**.

These choices of table-case mappings have a great impact on the amount of convergence and divergence that occurs, Section 6.2 presents more information on these choices and the

consequences they have. In the case studies presented in Chapter 9 we also show how different table-case mappings influence the event log and the process mining results. Furthermore, different sets of activities lead to different table-case mappings, for example, when only activities are chosen that are related to purchase requisitions, it is interesting to analyze these on a purchase requisition level instead of a purchase order level. The user should be able to make these decisions, i.e. (1) the activities to consider and (2) the table-case mapping to select, such that the focus of the process mining project can be set.

It is not always possible to find a case in an SAP process. Consider the example of a sales order, for which the items are not on stock and need to be procured (sketched in Figure 6.4). This process is very complex and can be seen as chain of several subprocesses. The process is roughly as follows: (1) the customer's sales order is received, (2) an item in the sales order needs to be procured from a vendor, (3) a purchase order is made for this item, (4) the purchase order is delivered to the warehouse, (5) the purchase order is billed (and paid), (6) the sales order processing is continued and the order is picked and packed, (7) the sales order is shipped and received by the customer and finally (8) the sales order is billed and paid. Here it is not possible to find one common case. There are however process models proposed to cope with complex processes like this; accompanied process mining techniques are now emerging that are able to deal with these kind of processes (see Section 6.3.1).

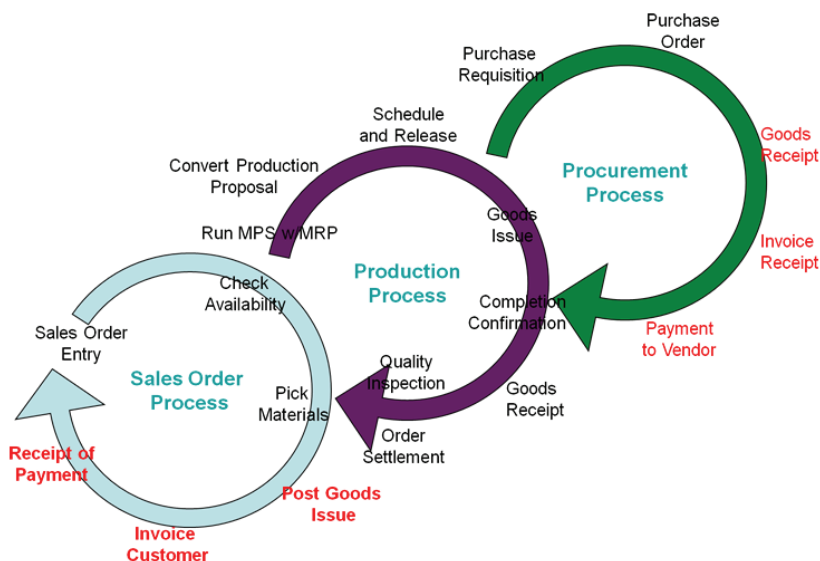


Figure 6.4: Integration of key SAP processes

6.2 Divergence and Convergence

The widespread adoption of database technology in (large) companies last century lead to the fact that developed information systems were often *data-centric*. These systems are still widely used, incorporated in the company and hard to get rid off. Creating a *process-centric* view for these systems is a difficult task and cannot be done without consequences. The

subsections below present two related issues frequently encountered when dealing with such data and proposes methods to deal with it. These issues should always be considered during the process mining phase and should be treated with care. Please note that the examples in these sections are simplified versions of how activity occurrences are actually detected in SAP, the main idea is however the same.

6.2.1 Divergence

As discussed in Section 2.2 one of the properties of an event log is that each event refers to a single process instance. We introduce the first of the two problems with an example, taken from our SAP IDES database. Table 6.6 presents a snapshot from the EKKO and BSEG tables.

Table 6.6: Example showing Divergence between Purchase Orders and Payments

EKKO: Purchasing Document Header		BSEG: Accounting Document Segment		
PO Number (EBELN)	Amount (NETPR)	Payment (BELNR)	PO Reference (EBELN)	Amount (WRBTR)
4500016644	82	5000000160	4500016644	32
4500013805	40	5000002812	4500016644	50
4500011015	30	4500011015	4500013805	40
		4500011015	4500011015	30

From the table above we can see that Purchase Order **4500016644** occurs two times in our BSEG table. The price of our Purchase Order amounts to €82, whereas it is paid in two terms with Payment **5000002812** for €50 and with Payment **5000000160** for €32. Now, what are the consequences of this? Suppose you would choose Purchase Order as case in the PTP process. For the process instance with case identifier 4500016644 we have **one** Create Purchase Order event, whereas we have **two** Payment events that are included in our event log. If no other events occur between these payment events, this results in *loops* in the process model. Most process mining algorithms do not specifically deal with this issue and visualize the multiple occurrences of the same activity in a process instance with a **self-loop**. If other events do occur in between such events the process model will become more complex. However, by **choosing a different case identifier**, this (problem) can often be solved. Let us reconsider our example from above and now analyse purchase orders on a lower level. Purchase Order Line Items are now included, Table 6.7 presents us the EKPO and (extended) BSEG table for the Purchase Order values from above.

Table 6.7: Example with Purchase Order Line Items and Payments

EKPO: Purchase Order Line Item		
PO Number (EBELN)	PO Item (EBELP)	Amount (NETPR)
4500016644	00010	50
4500016644	00020	32
4500013805	00010	40
4500011015	00010	30

When we now choose **Purchase Order Line Item** as case, each Purchase Order Line Item create activity has one related Payment activity in our example. Unfortunately, pur-

BSEG: Accounting Document Segment			
Payment (BELNR)	PO Ref. (EBELN)	PO Item Ref. (EBELP)	Amount (WRBTR)
5000000160	4500016644	00010	32
5000002812	4500016644	00020	50
4500011015	4500013805	00010	40
4500011015	4500011015	00010	30

chase order line items can still be payed in terms. This rarely happens; but our problem would thus be solved if each payment would only relate to one order line item.

The issue of the **same activity being performed several times for the same process instance** is entitled in [20, 4] the concept of **divergence** and is characterized as follows for event logs:

A divergent event log contains entries where the same activity is performed several times in one process instance. In a database structure, this is can be recognized by a n:1 relation from events to the process instance.

6.2.2 Convergence

The second of the two problems is also explained with the help of an example. Consider again the setting with Purchase Orders and Payments. What we can observe in Table 6.8 is that the Accounting Document with number **5000000164** contains two Accounting Document Line Items, both representing the payment of a different Purchase Order. This means that when this payment activity was executed, and the chosen case is the purchase order, *two payment events* would be created. All characteristics of this payment for both orders are exactly the same. During process mining analysis it would appear that a certain user was executing two payment activities at once. When it occurs on a larger scale in event logs this can have a big influence: the utilization of resources would not be reliable any more [4]. This also has an effect on characteristics such as the total number of payment activities executed and therefore on the total amount payed according to the event log. When we only look at purchase orders and want to retrieve the specific amount that was payed for that purchase order, we should *map the purchase order to the accounting document line item* as well. However, there is no relation between these fields, it cannot be decided how the payment is divided over the orders it corresponds to. These same problems occurs for purchase order line items, choosing another case has little influence on these issues.

Table 6.8: Example showing Convergence

EKKO: Purchasing Document Header	
PO Number (EBELN)	Amount (NETPR)
4500016000	132
4500013805	40
4500011015	30

The issue of the **same activity being performed in several different process instances** is entitled in [20, 4] the concept of **convergence** and is characterized as follows for

BSEG: Accounting Document Segment			
Payment (BELNR)	Payment Line Item (BUZEI)	PO Reference (EBELN)	Amount (WRBTR)
5000000164	001	4500016000	132
5000000164	002	4500013805	40
5000000171	001	4500011015	30

event logs:

A convergent event log contains entries where one activity is executed in several process instances at once. In a database structure, this can be recognized by a 1:n relation from an event to the process instance.

6.3 Ongoing Research

The upcoming section summarizes ongoing research related to the issues of convergence and divergence. In **process aware information systems** (PAIS), the problem of convergence and divergence can often be neglected. However, SAP's design, implemented based on objects and information is very data-centric and relies heavily on its underlying database. For these kind of systems, capturing a process in a structured monolithic workflow model is almost impossible. Section 6.3.1 presents an approach to deal with these kind of problems; it is very explorative and the effect on process mining is still being researched. In Section 6.3.2 we reflect these new possibilities on our approach.

6.3.1 Artifact-Centric Process Models

The use of *proclets* is advocated in [2] to deal with these kind of problems. As was observed in the previous sections, the different relations that exist between database entities (cardinalities 1:N, N:1 etc.) are a problem to cope with properly. Proclets aim to address these problems by representing processes as intertwined loosely-coupled object life-cycles, and making interaction between these life-cycles possible. Proclets were already introduced in the year 2000, however, renewed interest in tackling these problems, specifically the possibility of applying process mining on such models, leads to new research.

A proclet can be seen as a (lightweight) workflow process [2], able to interact with other proclets that may reside at different levels of aggregation. Recently, these kind of models have been referred to as **Artifact-Centric Process Models** [3]. Several distributed data objects, called **artifacts**, are present in such process models and are shared among several cases.

Current research at Eindhoven University of Technology by Fahland et al.[8] is investigating how process mining techniques can be applied on such models. A method is proposed to apply conformance checking on such models and (mining) plugins are developed for the ProM framework to support these models. An example of such an artifact-centric process model (taken from [8]) is given in Figure 6.5.

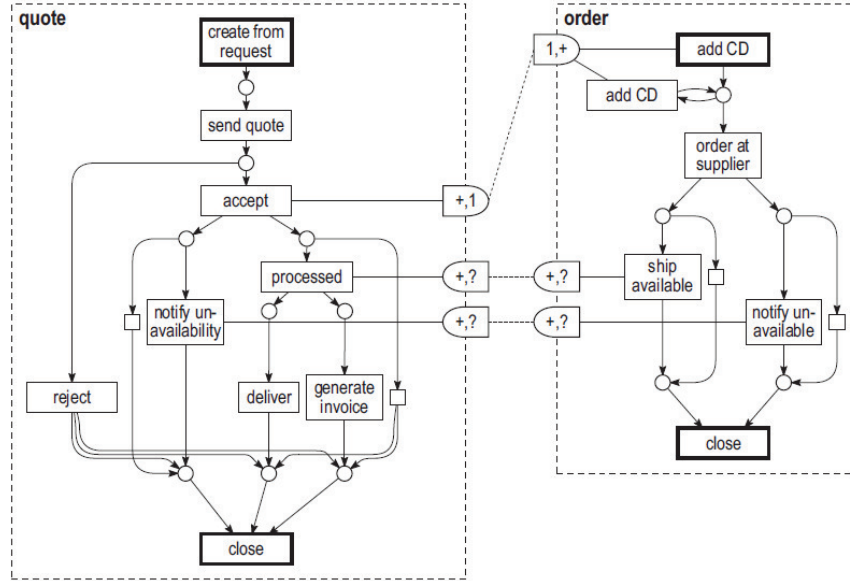


Figure 6.5: An artifact choreography describing the back-end process of CD online shop

In this example, the backend process of a CD online shop is considered in terms of pro-plets. From an artifact perspective, the artifacts *quotes* and *orders* can be identified. The decisive expressivity comes from the half-round shapes (ports), which have an accompanying annotation. The first part, *cardinality* specifies how many messages one artifact sends and receives to other instances, the second part, *multiplicity* specifies how frequent this port is used in the lifetime of an artifact instance.

More on these concepts and the example is explained in [8]. In the next section we discuss what possibilities there are when (workflow) processes are modeled as artifact-centric process models. More specifically, how can artifact-centric process models be used for process mining in data-centric ERP systems like SAP.

6.3.2 Possibilities for SAP

The previous section introduced the notion of artifact-centric process models. This section is explorative and discusses how these models could be applied in an SAP event log extraction process, regardless of the process mining software used. An important first step in implementing this approach is to (1) check whether *each activity can be mapped to an artifact*. For the PTP process this could be feasible. Imagine identifying the following artifacts in the PTP process:

1. Purchase Requisition
2. Purchase Order
3. Delivery
4. Invoice
5. Payment

(A Request for Quotations is a special type of Purchase Order and is therefore not mentioned in the above list)

In order to further support the artifact-centric approach, (2) new process models (pro-
cets) should be created that present the SAP processes and specify the interaction between
artifacts. (3) For each of these artifacts one could then specify life-cycles which capture the
activities related to that artifact. For the artifact Purchase Order we could for example have
the activities Create Purchase Order, Add Line Item, Delete Purchase Order, Close, etc. Fur-
thermore, (4) process mining software should be able to handle these new models in order to
apply (new) process mining techniques.

6.4 Conclusion

In this chapter we have presented an important part of this thesis: the determination of the
case in our event log extraction procedure. Event logs are structured around cases, the choice
of the case determines the view we eventually have on the process. We have presented a
method to propose possible cases for a given set of activities. These cases are represented in
the form of table-case mappings; a table-case mapping is a mapping of tables to a couple of
fields that together identify a case in that table. We have introduced issues that occur when
you focus on having one case notion in a process, and have presented current research that is
investigating how to tackle some of these problems.

Our table-case mappings are representations for cases that can be identified by different
fields in different tables. This approach is *not limited to SAP ERP systems*, but could be
applied to other ERP systems that rely on an underlying relational database as well. A pre-
condition for this is that the relations (foreign keys) between database tables are retrievable,
and that subsequent activities to other objects in a process can be traced back (linked) to pre-
vious objects (i.e. there is one central case that flows through the process). In our approach
we do not assume that specific SAP properties should hold, the approach can be generalized
to information systems that have an underlying relational database.

Convergence and divergence should always be taken into account in the process mining
phase. For data-centric ERP systems like SAP these issues are unavoidable, however, new
techniques are rising which are worth mentioning again. Artifact-centric process models show
good perspective on reducing issues that occur when performing process modeling and mining
for traditional data/object focused systems. However, research on this topic is still ongoing,
and mining algorithms and support in process mining software still has to be created. Future
research on process mining in SAP should therefore have a stronger focus on these issues, and
investigate the possibility of applying an artifact-centric approach to process modeling and
mining in SAP further.

Chapter 7

Incremental Updates

As mentioned in the research method presented in Section 1.3, one of the goals of this project is to develop a method to incrementally update a previously extracted event log from SAP. This should be done with only the changes from the SAP system that were registered since the original event log was created.

At the time of performing this Master’s project, few research was done in this area. The incremental aspect in most of that research is at a process model level. With this we mean that methods are proposed to incrementally update process models with new data. For example, in [22] an incremental workflow mining algorithm is proposed, based on intermediate relationships in the workflow model such as ordering and independence. However, the data could be such that the updated process model would be completely different than discovering the process model with the entire (updated) data. In our project we do not focus on updating at the process model level, but focus on incremental updating at the *event log level*. This updating of event logs can be seen as extending existing event logs.

The most important benefit of being able to update an event log is that changes within a process can be discovered quicker. Of course one could simply extract the entire event log from scratch to reach that same goal, but for large event logs, consisting of hundreds of thousands of events, updating an event log is much more beneficial.

This chapter starts off by presenting an overview of our event log update approach (Section 7.1), in which timestamps play an important role. It includes the assumptions and decisions we make, as well as some issues that should be considered in order to get our approach to work. The procedure to actually incrementally update a previously extracted event log is presented in Section 7.2, where the various steps are outlined in the accompanied subsections. Section 7.3 concludes this chapter by recapitulating everything that is discussed and addressing if SAP is really suitable for incremental updating of event logs.

7.1 Overview

In this section we present an overview of our **timestamp** approach to update event logs. This is schematically explained through Figure 7.1. The timestamps are represented by t_0 , t_1 , t_2 and t_3 . The data that contains events that occurred between t_0 and t_1 is represented by

D_0 , between t_1 and t_2 by D_1 and between t_2 and t_3 by D_2 . This implies that the data that covers events that occurred between t_0 and t_3 is found in $D_0 + D_1 + D_2$. The database in which we store this data thus contains different data depending on the timestamp till which it is up to date.

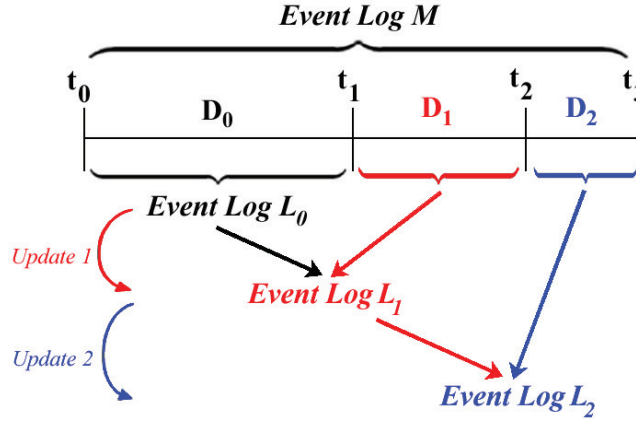


Figure 7.1: Working with Timestamps

In practice: if we perform a normal event log extraction (as described in Chapter 5) from data $D_0 + D_1 + D_2$, we retrieve all events that occurred between t_0 and t_3 in event log M . If we extract an event log L_0 from data D_0 , subsequently update this D_0 with data D_1 , and update this event log with events that occurred between t_1 and t_2 we get event log L_1 . If we then continue this (i.e. the incremental aspect) with data D_2 , extract all events that occurred between t_2 and t_3 and write this to an event log L_2 , the resulting event log L_2 should equal event log M ; that is: contain exactly the same events ($M \equiv L_2$).

Summarizing, we can define a correct update of an event log with the following goal:

Goal: An update of an event log L_0 that was extracted with data D_0 , to an event log L_1 , using update data D_1 , should lead to the same event log as when extracting a new event log M with data $D_0 + D_1$, i.e. $L_1 \equiv M$.

Figure 7.1 thus describes *two* incremental updates of an event log L_0 . This procedure can be prolonged each time new data is available (i.e. $D_3, D_4, \dots$). Furthermore, in practice we do not maintain three separate event logs (L_0, L_1, L_2); we append the ‘new events’ to the original log (L_0), therefore extending it. This approach assumes that, when we for example update data D_0 with data D_1 , the addition of D_1 does not lead to newly generated events from D_0 , as well as that no events are removed from D_0 . Below we reformulate this assumption and present another assumption and two implementations decision that support the timestamp approach.

7.1.1 Assumptions

The section above clarified that we have to assume that events in an event log (and thus the data) are bound to one certain time interval. If we update a database with new data, we should not be able retrieve new events from that old time interval.

A1 *An event is bound to a time interval.*

A second assumption we have to make results from the table-case mapping approach. It is given below; if this does not hold, we could possibly not relate events that handle the same case through their case identifier.

A2 *The Primary Key fields in the SAP database, as well as their values, are not changed.*

7.1.2 Decisions

We further have to make two (implementation) decisions in order to be able to perform a correct (incremental) update of an event log, and deal with all the issues that were presented in Section 7.1.3.

D1 *When a database update is performed, it is updated up to a certain timestamp. That is, one can assume that each table is up to date up to the same timestamp.*

D2 *An event log update is always performed based on the last extraction timestamp (or update timestamp) known for that event log.*

Both decisions actually follow from Figure 7.1. **D1** ensures that updating the local database with new data results in an update of **all tables** to the same timestamp. **D2** indirectly implies that an event log is up to date to the timestamp the local database was up to date to at the time of extraction (or update).

7.1.3 Exploration

Before we can achieve our goal and propose a procedure to update event logs we first explore some concepts that should be considered in order to avoid erroneously constructed event logs. An event log is a structured file and an event log update should correctly extend the event log with new events.

- **Case Selection:** the case instance that accompanies each event ensures the grouping of events that belong to the same case. When updating an event log, all added events should therefore have the same notion of a case (e.g. not *Purchase Order* in the original event log and *Payment* in the added events). This means that the same table-case mapping as in the original event log should be used during an update of this event log.
- **Duplicates:** ensure that the updated event log does not contain duplicate events. When performing an event log update, events that were extracted before should not be considered anymore. We somehow have to ‘memorize’ or filter those previously extracted events.

- **Timestamps:** incrementally updating of event logs is strongly bound to the notion of time. Each table has many date and time fields, one has to ensure the correct *Created On* or *Changed On* timestamps can be identified.
- **Incrementally Updating:** continuously updating an event log should not lead to additional problems.

All these issues follow from our goal and can be summarized into a notion of **soundness** and **completeness**: an update of an event log should result in the same number of events in that event log as when performing an entire event log extraction from scratch. More specifically, we should have exactly the same events in both updated and normally extracted event log, only the *order* in the file might differ.

7.2 Update Procedure

We now propose a procedure to update a previously extracted event log that is driven on our assumptions and implementation decisions and considers the concepts explored. This procedure is given in Figure 7.2.

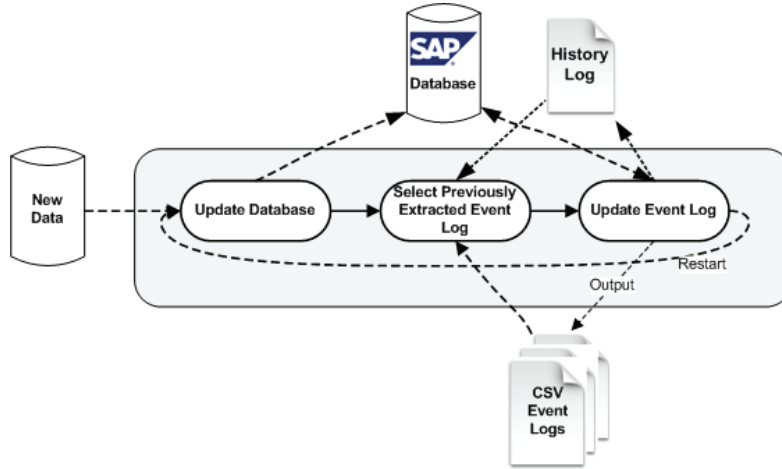


Figure 7.2: Update Procedure

In order to perform an event log update, we first need new data. The first step is therefore to ensure that we have the latest version of the SAP database at our disposal. The SAP database in the figure again represents a local copy of the SAP database. In the procedure the update is done in step (1) *Update Database*. Having updates available, the next step is to (2) *select a previously extracted event log* on which we perform our update. The most important step is the final step: (3) the actual *update of the event log*. The incremental aspect is represented by the loop, meaning that updates can be performed repeatedly, requiring the presence of new data (downloaded from the actual SAP database) at the start of each loop in order to make sense. Below we discuss these three steps in more detail; in Section 8.2.2 we elaborate on how these actions are actually implemented in our application prototype.

7.2.1 Update Database

Looking from a more general perspective, this step can be seen as ensuring we have the latest version of the SAP database at our disposal. One could assume that we always have the latest version in our local database; however, we have to ensure this database can be brought up to date. Suppose we have a set of tables T that contain the data with which we want to update our database DB , the algorithm to update the database is as follows:

1. **For each** table t_{new} in the set T
2. $t :=$ target table in DB
3. **Insert** t_{new} **into** t

7.2.2 Select Previously Extracted Event Log

By selecting a previously extracted event log we know the timestamp of the original extraction and find out the case that was used in the event log. This last thing is very important since otherwise we would not know how to identify cases within our new data, and thus relate events.

7.2.3 Update Event Log

The last step in this procedure, the actual updating of the event log, is similar to our *Constructing the Event Log* step from Figure 5.1. We now have to make sure we only extract the events that occurred within a given timestamp interval. Furthermore, the actual updating of the CSV event log file is smoothened by Futura Reflect's event log format. This format, and the way Reflect handles it, does not require that events that handle the same case are grouped or even chronologically ordered, we can just append new events to the end of the event log.

We now present the actual algorithm to update a previously extracted event log. It is very similar to the algorithm presented in Section 5.4. Suppose A is the set of activities we want to extract and L the event log we want to update, updating this event log can be performed with the following algorithm:

1. **Extract** table-case mapping for L
2. **Retrieve** timestamp information t for L
3. **For each** activity $a \in A$
4. Retrieve occurrences of a that happened after t , store results in R
5. **For each** record $r \in R$
6. Extract attributes att from r
7. Append case identifier for r and att to L

With *extracting the table-case mapping* in line 1 we mean that we retrieve how cases are represented in the existing event log (e.g. with fields like MANDT, EBELN, EBELP for activities that have table EKPO as 'base table'). This ensures that cases are represented in the same way throughout the updated event log. In Line 2 we retrieve when the event log L

was extracted. This enables us to set constraints that ensure that only events are retrieved (line 4) that occurred after a specific timestamp (after t).

7.3 Conclusion

This chapter has shown that incrementally updating a previously extracted event log from SAP is feasible, given that the timestamp approach can be implemented. We schematically introduced our timestamp approach in Section 7.1; this included a goal that defined when an incremental update is correctly performed, as well as two assumptions and implementation decisions that should be made in order to correctly perform such an update. After that we presented the procedure to perform incremental updates of event logs and discussed the various steps.

Chapter 8 presents our prototype, including the implementation of the incremental update procedure. Normally, if you would continuously update an event log with new data, one would think that more events could be detected because we are monitoring the data at multiple points in time. However, our timestamp approach states that this should not make a difference. A precondition for this is that the approach can successfully be implemented with SAP. It is promising because, in SAP we know that each base table contains a Changed On and Created On field which eases the retrieval of new records. The Change Tables do not seem to pose problems as well: each record holds information about one event, the recorded timestamps allow for splitting of event occurrences between certain timestamps.

Chapter 8

Prototype Implementation

Chapter 5 started off by presenting a simple flow diagram that showed our procedure of extracting an event log in SAP. Technical details were avoided so far; this chapter continues with the same flow diagram from Chapter 5, extends it and introduces a prototype that operates within this procedure. This application prototype implements the method of case determination as presented in Chapter 6 and supports the incremental updating of event logs as described in Chapter 7.

In this chapter we first of all present the extended flow diagram in which the prototype is embedded in Section 8.1. The various components out of which this flow diagram consists are explained in the accompanying subsections. Our prototype enables the incrementally updating of event logs; because this was not yet introduced within our extraction procedure from Chapter 5, we introduce this functionality as an extension of that procedure (see Section 8.2). Section 8.3 delves deeper into the technical details behind the development and architecture of our prototype. In Section 8.4 we give a graphical introduction to our prototype with some screenshots, covering all important functionality. Section 8.5 lists some improvements that can be made to our prototype, especially to further smoothen the incremental updating of event logs. In Section 8.6 we draw our conclusion about the implementation.

8.1 Overview

The process in Figure 8.1 is an extension of Figure 5.1. The preparation and extraction phase can again be identified; this separates what has to be configured once for each process from the actions in the prototype that can be done repeatedly. We discuss this diagram by splitting it in two parts: **(1)** creating the process repository (i.e. preparation phase, Section 8.1.1) and **(2)** external interfaces (SAP and Futura Reflect, Section 8.1.2). The prototype itself is not discussed in detail. The four main steps within the prototype concern user actions that need to be done through the GUI (i.e. *Selecting Activities to Extract* and *Selecting the Case*, see Section 8.4) or are implementations of previously mentioned steps. For the computation of the *Table-Case Mappings* we refer to Chapter 6; the actual construction of the event log was introduced in Section 5.4.

Compared with Figure 5.1 we see an addition of the step *Extracting Foreign Key Relations* in the preparation phase. This step is necessary to enable the computing of table-case

mappings later on. The extraction phase is extended with two steps, *Selecting Activities to Extract* and *Computing Table-Case Mappings*, to enable the user to specify its own variation of the concerned business process.

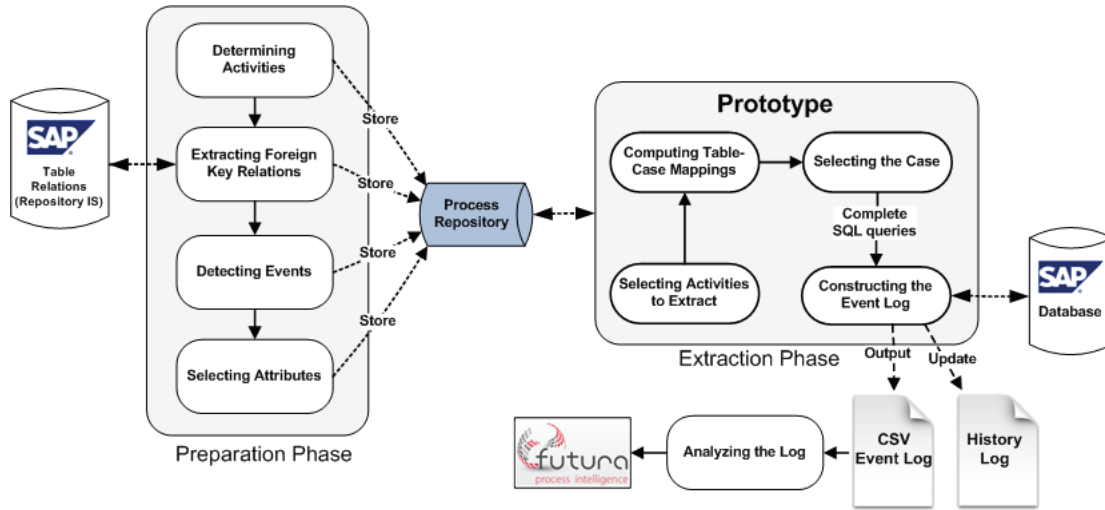


Figure 8.1: Extraction Procedure with Prototype Included

8.1.1 Preparation Phase

One of the main goals of our prototype is to smoothen the event log extraction for SAP processes. More specifically: once all required information for event log extraction for a given business process is gathered and stored as defaults, event logs for that process should be able to be extracted repeatedly with these stored defaults. The first steps in our event log extraction procedure (*Determining Activities*, *Extracting Foreign Key Relations*, *Detecting Events* and *Selecting Attributes*) therefore ensure the creation of a **repository** that holds all information regarding processes, activities in processes and relations between tables (activities). This *repository* should be created for each process.

In this repository we maintain a couple of CSV files that can be configured and hold information about various aspects of that process. The combination of such files for one process is what we call **Process Repository**. The user should **create and configure these files**, the prototype does not provide an interface for that. However, this step only needs to be performed *once* for each *new SAP process* that is not yet included in the prototype. Information from these process repositories can be reused immediately, allowing a user to *repeatedly* extract an event log for the same process.

Process Repository Overview

Configuration of the prototype is thus mainly done through CSV files at the moment. A similar repository could be created in a database format, but this is not considered in this project. Table 8.1 gives an overview of all files that need to be created and configured per process in order to perform an event log extraction for that process. The upcoming subsections discuss their structure and in which step they are created.

Table 8.1: CSV Configuration Files

File Name	Description
<ProcessName>activitiesToTables.csv	Lists how to set up SQL queries for occurrences of each activity.
<ProcessName>relations.csv	Lists all foreign key relations for tables involved in the process.
<ProcessName>keyAttributes.csv	Lists executor and timestamp (created on) fields for each table occurring in <ProcessName>activitiesToTables.csv.
<ProcessName>attributes.csv	Lists all additional (interesting) attributes for each table occurring in <ProcessName>activitiesToTables.csv.
<ProcessName>tableTitles.csv	Lists the textual description of each table.

Determining Activities

Section 5.3.1 describes various approaches to gather activities that exist in an SAP process, and Section 6.1 explains how we could retrieve the (base) tables that correspond to these activities. This information is combined and stored in CSV format in our process repository in a file called <ProcessName>activitiesToTables.csv, where for each activity we store the related base table. The first lines of the file PTPactivitiesToTables.csv are given in Listing 8.1, where the format of each line is as follows: <Activity>;<Base table>.

```
Create Purchase Requisition;EBAN
Change Purchase Requisition;EBAN
Delete Purchase Requisition;EBAN
```

Listing 8.1: Excerpt of the PTPactivitiesToTables.csv file

Extracting Foreign Key Relations

Furthermore, we need to store information about the relations that exist between the identified tables (including lookup tables) in our repository. Acquiring these (foreign key) relations from SAP is described in Section 6.1 as well, and is done through SAP's Repository Information System. The format that describes each foreign key is the same as SAP uses, an extra column is added to distinguish between foreign keys. For each table involved in a process we store **all foreign key relations** in a file called <ProcessName>relations.csv; Listing 8.2 presents an excerpt of the file PTPrelations.csv.

```
T000;MANDT;CDHDR;MANDANT;N
TSTC;TCODE;CDHDR;TCODE;N
T161;MANDT;EBAN;MANDT;N
T161;BSTYP;EBAN;BSTYP;
T161;BSART;EBAN;BSART;
T024;MANDT;EBAN;MANDT;N
T024;EKGRP;EBAN;EKGRP;
```

Listing 8.2: Excerpt of the PTPrelations.csv file

The structure of each line is as follows: <Check table>;<Check table field>;<Foreign key table>;<Foreign key field>;<New foreign key indicator>. A foreign key is composed of a (number of) line(s). More specifically, the first line of a foreign key is indicated with an 'N' in the last *column*, all lines below that line, until a line that again has an 'N' in the last column, belong to the same foreign key. In the file above we can for example find four foreign keys. For the third foreign key, in the *foreign key table* EBAN, the fields (MANDT, BSTYP, BSART) are related to the primary key fields (MANDT, BSTYP, BSART) of table T161 (*check table*).

Detecting Events - Setting up Base SQL Queries

To construct SQL queries for activities, we need the information that is gathered by following the approach proposed in Section 5.3.2. This information typically consists of a table name, column values through which the activity can be identified, lookup tables etc. The goal is thus to construct these SQL queries and store them in our process repository. The queries should enable us to retrieve occurrences of certain activities. Experience with SQL is needed in order to set this up, but SQL, as the standard querying language for relational databases, is widely familiar these days and known by the people this graduation project targets at.

For example, we know that creating a Purchase Requisition results in a new record (exactly one) in the table EBAN. To retrieve all occurrences of the activity **Create Purchase Requisition** (i.e. events that concern this activity) we only have to perform the following SQL query:

```
SELECT * from EBAN
```

Our prototype combines this SQL query with the table-case mapping that is chosen. This means that from the returned records, we select the fields that represent the case for that query (i.e. accompanied table). If a case on *purchase requisition level* is chosen (e.g. a table-case mapping that is calculated for events Create Purchase Requisition, Change Purchase Requisition, Delete Purchase Requisition), the combination of MANDT (Client), BANFN (Purchase Requisition Number) and BNFPO (Purchase Requisition Item) represents a case. On the other hand, when more activities are involved (i.e. activities related to Purchase Orders), a case could be chosen that is represented by the combination of MANDT, EBELN (Purchasing Document Number) and EBELP (Purchase Order Line Item). In this case we would only select Purchase Requisitions that refer to a purchase order. In our example this can be done since purchase requisitions hold references to purchase orders in EBAN through the EBELN and EBELP fields. When there is no reference, these fields are empty. So, due to the fact that purchase orders not always refer to purchase requisitions and vice versa, the results of the example query above should be handled in different ways depending on the table-case mapping that is chosen. The prototype thus supports one type of SQL query per activity, but interprets the query results differently based on the table-case mapping selected.

Querying the *change tables* is a bit more difficult than querying regular tables. As mentioned in Section 4.2.1 and 5.3.2, the link from an event in the change table to the record in their base table is done through column *TABKEY* in CDPOS. The format of the values in *TABKEY* may differ from event to event, that is, from table to table. A change to a purchase

requisition with *MANDT* = 090, *BANFN* = 0010000992 and *BNFPO* = 00010 has TABKEY 090001000099200010, whereas a change in for example shipping notification with *VBELN* = 0180000107, *POSNR* = 000004 and *MANDT* = 800 has TABKEY 8000180000107000004. The number of characters that are reserved can therefore differ, but mostly relates to the primary key of the related table (TABNAME in CDPOS). Thus, when events should be detected through the change tables, it is important to be able to deduce the case representation from the accompanied TABKEY.

In order to deal with all these different scenarios and support the idea of being able to chose different cases, our process repository is extended with a mapping between activities and SQL queries. The <ProcessName>activitiesToTables.csv file presented earlier is extended to include information that is necessary to build up the SQL query. An example of this renewed file can be found in Listing 8.3.

```

1 Create Purchase Requisition;EBAN;;1;SQL;*;EBAN;TRUE;
2 Change Purchase Requisition;EBAN;;1;CHANGE;USERNAME,UPDATE,UTIME;MANDT,3#
  BANFN,10#BNFPO,5;TABNAME='EBAN' AND FNAME<>'LOEKZ';
3 Delete Purchase Requisition;EBAN;;1;CHANGE;USERNAME,UPDATE,UTIME;MANDT,3#
  BANFN,10#BNFPO,5;TABNAME='EBAN' AND FNAME='LOEKZ' AND VALUE_NEW='X'
  AND VALUE_OLD='';
4 Undelete Purchase Requisition;EBAN;;1;CHANGE;USERNAME,UPDATE,UTIME;MANDT,3#
  BANFN,10#BNFPO,5;TABNAME='EBAN' AND FNAME='LOEKZ' AND VALUE_NEW='' AND
  VALUE_OLD='X';
5 Change Request for Quotation;EKPO;EKKO;1;SPLIT;*;CDPOS, CDHDR, EKKO, EKPO;
  TABNAME='EKPO' AND FNAME<>'LOEKZ' and CDPOS.changenr = CDHDR.changenr
  and substring(TABKEY from 4 for 10) = EKPO.anfnr and EKPO.ebeln = EKKO
  .ebeln and EKKO.bstyp = 'A';MANDT,3#EBELN,10#EBELP,5;
```

Listing 8.3: Excerpt of the PTPactivitiesToTables.csv file

For each activity we have one line in this file. The first column indicates the name of the activity, the second column the base table for the activity, the third column a possible lookup column (like BKPF for BSEG), the fourth column indicates if the activity should be shown in the prototype (1 = yes, 0 = no) and the remaining columns contain information necessary to compose the SQL query. The method to do this differs per activity.

SQL

A simple SQL query is indicated with *SQL* in the fifth column. The accompanying query is constructed from the remaining three columns, that respectively represent the SELECT, FROM and WHERE clauses.

CHANGE

Querying for activity occurrences that need to be retrieved from the change tables, denoted by *CHANGE* in the fifth column, is done in a different manner. These ‘change table activities’ are accompanied with some key attribute fields in the sixth column, an identifier that specifies the structure of the previously mentioned TABKEY (e.g. MANDT,3#BANFN,10#BNFPO,5) in the seventh column (to link it to a case) and a WHERE clause in the last column. The

prototype automatically completes the select, from and where clause for the query such that the CDPOS and CDHDR tables are used and joined.

SPLIT

A third possibility concerns activity occurrences that are retrieved from the change tables as well, however, more information than just from the change tables is required to create the events. These activities are denoted by a *SPLIT* value in the fifth column of our CSV file. One can think of activities where retrieved change table records have TABKEYs that cannot directly be linked to case (i.e. it needs to be looked up in another table). Here the sixth, seventh and eighth column respectively represent the SELECT, FROM and WHERE clause of the SQL query. The prototype further specifies this query with the ninth column, that creates the link between the TABKEY and a record in the base table.

Having this three classes, this means that the prototype is thus not fed directly with a set of queries that can be executed at once in a target database. The SQL queries are completed within the prototype later on, based on the three ‘activity classes’ above. There are also separate routines for each of the three activity classes above to process the query results.

Selecting Attributes

Besides the CSV files mentioned so far, our process repository holds information about what attributes need to be selected for each activity. First of all, the **timestamp** and **executor** of an event needs to be present in an event log. Presence of timestamps for events in an event log is mandatory when you want to discover the control-flow with process mining. This determines the order of events/activities in the process. The executor of the event is another attribute that needs to be present: when constructing a social network this attribute is indispensable. We specify the timestamp and executor fields for each table in a file called `<ProcessName>keyAttributes.csv`, for the PTP process, a part of that file is as follows:

```

1 EBAN;ERNAM;BADAT;;;
2 EKBE;ERNAM;CPUDT;CPUTM;;
3 LIPS;ERNAM;ERDAT;ERZET;;
4 MSEG;USNAM;CPUDT;CPUTM;MKPF;MANDT,MBLNR,MJAHR
5 RSEG;USNAM;CPUDT;CPUTM;RBKP;MANDT,BELNR,GJAHR
```

Listing 8.4: Excerpt of the PTPkeyAttributes.csv file

Each line has the following structure: `<Table>;<Resource>;<Date>;<Time>; <LookupTable>;<Link Through>`. In Listing 8.4 we can observe three different types of lines. **(1)** lines (e.g. line 1) that do not contain a time field; unfortunately it is indeed possible in SAP that an exact time for an event can not be retrieved, in this case only the date is used by the prototype, using a time of `00:00:00`. **(2)** Line 2 and 3 concern tables for which we can retrieve timestamp and resource information directly from that table. **(3)** Line 4 and 5 deserve a bit more attention. Because activities are linked to base tables, our prototype queries the `<ProcessName>keyAttributes.csv` file using that base table. If a base table however does not contain timestamp and resource information, but if it can be looked up in a header table, then the fifth column of the file specifies the *lookup table*. The base table and lookup table are then linked with fields present in the sixth column (the field names are the same for both

tables), the timestamp and resource fields for that lookup table are still specified in column two and three.

Additional Attributes

An event log can be accompanied with additional attributes that aid in the analysis of the mined process later on. These additional attributes that should be written to the event log are specified in the file `<ProcessName>attributes.csv`. This file is not compulsory, an example of some lines in such a file for the PTP process is given below in Listing 8.5.

```
1 EBAN;Material Number;MATNR;1;;
2 EBAN;Purchase Requisition Quantity;MENGE;2;;
3 EBAN;Purchasing Group;EKGRP;1;T024;EKNAME
4 EKPO;Short Text;TXZ01;1;;
5 EKPO;Plant;WERKS;1;T001W;NAME1
6 EKPO;Company Code;BUKRS;1;T001;BUTXT
```

Listing 8.5: Excerpt of the PTPattributes.csv file

Each line has the following structure: `<Table>;<Description>;<Field>;<Use>;<Lookup table>; <Lookup column>`. For each table we specify a number of interesting attributes that should be included in the event log. In our prototype, when activity occurrences are queried, the accompanied base tables in `<ProcessName>attributes.csv` specify which additional attributes should exactly be included.

We can again observe a classification of lines. **(1)** Lines that only specify the table, the field that contains the attribute and a description of the attribute (to include in the first line of the event log later on). **(2)** Some attributes are rather cryptical and only contain codes that are difficult to interpret. Columns five and six (when filled in) allow for retrieving the value accompanied with such a field (in column three) from a lookup table. For example, the *purchasing group* attribute in *EBAN* is specified by field *EKGRP*, this is a number (e.g. 854), the name of the purchasing group needs to be looked up in table *T024* and can be found in field *EKNAME* (e.g. Brisbane). The field *EKGRP* serves as the link between both tables, the field name is in both tables the same.

TableTitles

Another CSV file that needs to be created is a file that holds textual descriptions of tables. It aids the user of the prototype by returning these names with each table name. It has to be created for each process, contains the tables that are used in this process and has the following name: `<ProcessName>tableTitles.csv`. An example of this file for the PTP process is found below, the structure of each lines is as follows: `<Table>;<Description>`.

```
BKPF;Accounting Document Header
BSEG;Accounting Document Segment
EBAN;Purchase Requisition
EKBE;History per Purchasing Document
```

Listing 8.6: Excerpt of the PTPtableTitles.csv file

History Log

Followed from the sections above, an important addition to our process repository concerns the creation of *event log awareness*. This is achieved by having one **history log** file that stores information about all previously extracted event logs. An excerpt of this file, `historyLog.csv`, is given in Listing 8.7.

```

1  2011-02-16;02:14:29;OTC 16-02-2011 02.18.03.csv;n/a;n/a;OTC;MSEG#[MANDT,
    KDAUF, KDPOS]@VBAP#[MANDT, VBELN, POSNR]@VBRP#[MANDT, VGBEL, VGPOS]
    @VBUP#[MANDT, VBELN, POSNR]@LIPS#[MANDT, VGBEL, VGPOS]@VBFA#[MANDT,
    VBELV, POSNV]
2  2011-02-23;09:44:50;PTP 23-02-2011 09.19.42.csv;n/a;n/a;PTP;BSEG#[MANDT,
    EBELN, EBELP]@EKBE#[MANDT, EBELN, EBELP]@MSEG#[MANDT, EBELN, EBELP]
    @EBAN#[MANDT, EBELN, EBELP]@LIPS#[MANDT, VGBEL, VGPOS]@EKPO#[MANDT,
    EBELN, EBELP]@RSEG#[MANDT, EBELN, EBELP]@EKKO#[MANDT, EBELN, LPONR]
3  2011-02-23;10:39:47;PTP 23-02-2011 10.35.21.csv;2011-02-25;15:18:15;PTP;
    BSEG#[MANDT, EBELN, EBELP]@EKBE#[MANDT, EBELN, EBELP]@MSEG#[MANDT,
    EBELN, EBELP]@EBAN#[MANDT, EBELN, EBELP]@LIPS#[MANDT, VGBEL, VGPOS]
    @EKPO#[MANDT, EBELN, EBELP]@RSEG#[MANDT, EBELN, EBELP]@EKKO#[MANDT,
    EBELN, LPONR];
4  2011-02-25;16:01:56;PTP 25-02-2011 03.57.04.csv;n/a;n/a;PTP;EKBE#[MANDT,
    EBELN]@BSEG#[MANDT, EBELN]@MSEG#[MANDT, EBELN]@EBAN#[MANDT, EBELN]@EKPO
    #[MANDT, EBELN]@LIPS#[MANDT, VGBEL]@RSEG#[MANDT, EBELN]@EKKO#[MANDT,
    EBELN]
```

Listing 8.7: Excerpt of the History Log

In total we can identify seven fields in each line of the CSV file, the lines are structured as follows: <Extraction Date>;<Extraction Time>;<Event Log File Name>;<Update Date>;<Update Time>;<Process Name>;<Table-Case Mapping>. The activities that were selected in the extraction of an event log are not stored currently. So, reflecting the meanings of these fields on Listing 8.7. Line 1 concerns an event log extracted for the OTC process on 2011-02-16 02:14:29. The other three lines concern the PTP process; from line three we can for example conclude that the file *PTP 23-02-2011 10.35.21.csv* is updated two days after the extraction at 15:18:15. Furthermore in line four the stored table-case mapping consist of fewer fields than the others, in this case indicating that a table-case mapping on Purchase Order level was chosen.

8.1.2 External Interfaces

Our prototype communicates internally with the process repository. We can characterize the communication with SAP and Reflect as external communication.

Communication with SAP

Besides extracting foreign key relations from SAP, or consulting SAP in an informative way (e.g. how to detect activity occurrences), we have to execute SQL queries on the underlying SAP database to acquire the necessary data to put in our event log. Currently, our prototype

does not communicate directly with SAP for this. A local copy of the relevant tables in our SAP IDES database is made in *PostgreSQL* using the approach presented in Section 4.2. This is first of all beneficial for testing purposes, another thing is that companies often do not allow direct communication with their data/database.

We first used plain CSV files to represent our SAP IDES database (tables can be extracted in this format from SAP), but this soon became too complex and slow to query. There exist drivers to query a collection of CSV files as if they would represent a relational database (e.g. StelsCSV¹), however, performance- and license wise this idea was set aside and a local copy of the SAP IDES database in *PostgreSQL* was created and used.

There exist methods to synchronize a RDBMS with the SAP database, but this is not investigated in this project. The Java Connector presented in Section 4.1.1 could for example be integrated in our prototype such that it communicates with SAP by means of RFC's. Data can then be retrieved and updated in a (local) database. Another possibility could be to execute the SQL query directly into the SAP system, but all this requires much more investigation.

Futura Reflect

The event logs our prototype outputs adhere to the event log format supported by Futura Reflect. Event logs are stored as CSV files. Each line in the CSV file represents an event; the values at each line are delimited by a delimiter (e.g. a comma or semi-colon) and can contain an arbitrary number of values. These values represent the attributes of our event log. The order of the attributes in a line are not fixed, but must be the same for each line. Semantics is given to the attributes when importing it in Reflect. Although auto-detect functionality of attribute formats is becoming more advanced, it is useful to have insight in the structure of the event log. Our prototype supports this by including descriptions of each event field in the first line of the event log, however, it is for example still to the user to decide if an attribute should be considered on a case or event level.

```

1 13966,2009-01-17 00:00:00,Goods issue,HAMED,4500009353,,10,,,,,,,,,552.00
2 13967,2009-09-23 00:00:00,Request requisition,JJANS,0010012461,Purch.
   requis. Stand.,10,,,,,IDES Deutschland,,,0.00
3 13967,2009-09-23 00:00:00,Create requisition,USERADMIN,0010012461,Purch.
   requis. Stand.,10,,,,,IDES Deutschland,,,0.00
4 13967,2009-09-23 00:00:00,Release requisition,JJANS,0010012461,Purch.
   requis. Stand.,10,,,,,IDES Deutschland,,,0.00
5 13968,2009-11-26 00:00:00,Request requisition,JJANS
   ,0010002943,,10,,,,,,,,,0.00
6 13968,2009-11-26 00:00:00,Release requisition,JJANS
   ,0010002943,,10,,,,,,,,,0.00

```

Listing 8.8: Excerpt of a CSV Event Log

¹<http://www.csv-jdbc.com/>

Consider the example in Listing 8.8. In this example the format of each line is as follows: `<Case Identifier>, <Timestamp>, <Activity Name>, <Resource>, <Case Attribute 1>, <Activity Type>, ..., <Additional attributes>`. When importing this event log in Reflect you have to indicate which column denotes the case identifier, the activity, the accompanied event timestamp etc. Furthermore you have to specify the format for each attribute, e.g. if it is a text value, integer or something else. In the example, lines that belong to the same case identifier are grouped (e.g. for case identifier 13967). This is not required however, each line should contain an event, a sequence of lines (events) does not have an other meaning than if these lines (events) would have been spread throughout the CSV file. This means that events in the event log should not be chronologically ordered or grouped per case. Each line could thus belong to a different case identifier, Reflect groups events that have the same case identifier upon importing the file.

These plain CSV text files can have an arbitrary length; Reflect is adapted to cope with such large event logs. Furthermore, the CSV event log format is pretty flexible and close to logging formats used within companies, which requires few adaptations to existing logs in order to transform it to a CSV event log.

8.2 Incremental Updates

We introduce our incremental update support as an addition to our basic event log extraction procedure. Section 8.2.1 first shows how this event log update procedure can be embedded in the prototype. Section 8.2.2 discusses all extensions that have to be made to our prototype to support the incremental updating of event logs, more specifically the changes to our process repository.

8.2.1 Overview

In Figure 8.2 we can find the merge of two *flow diagrams* (Figure 7.2 and 8.1). Besides the preparation and extraction phase, we now see the addition of an *update phase*. The steps in this phase refer to the steps presented in Section 7.2. This starts with *Update Database*, which updates our local copy of the SAP database with *new data*. As explained in Section 7.2.1, this will bring our local database up to date to a certain timestamp. This step can be omitted if our prototype would have a direct communication link with the SAP database and is able to automatically access the latest data. However, because the prototype is linked to the local database we provide support to update this local database ourselves with new data. Another step that might require some explanation is *Update Event Log*. Our prototype implements the procedure from Section 7.2.3 and appends new extracted events to an existing event log. The upcoming section present the implementation details behind this step; the update phase can be restarted again when new data is available.

8.2.2 Prototype Extensions

As we assumed in Section 7.1.1, our database is always up to date to a certain timestamp, say t_1 . When we extract an event log, we thus have extracted all events till timestamp t_1 . An update of the database results in the database being up to date till timestamp t_2 . Our goal is to find those events that occurred between timestamp t_1 and t_2 and add them to our event

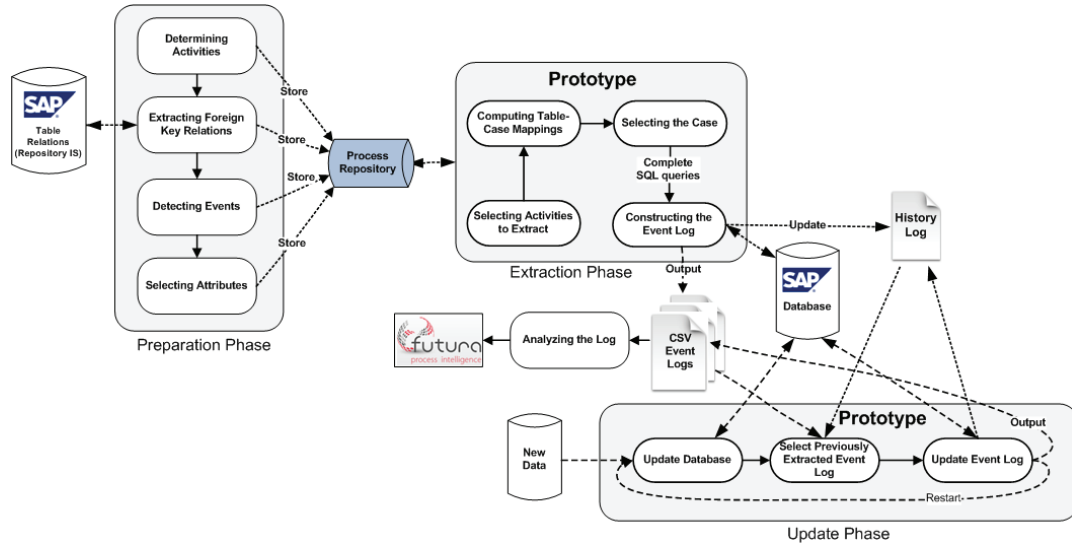


Figure 8.2: Extraction Procedure with Update Loop

log. It is clear that **timestamps** of events play a very important role. These timestamps t_1 (**event log extraction date**) and t_2 (**database updated to date**) should however be used differently per type of activity. The first addition we have to make to our process repository are new SQL queries to support in finding these events. Consider again the three activity types presented in Section 8.1.1: SQL, CHANGE and SPLIT.

CHANGE

Activities in the class *CHANGE* are activities whose occurrences should solely be retrieved through the change tables. The change tables log the date and time when a change occurred. So in order to retrieve events that occurred after our initial event log extraction (t_1), we have to extend our SQL query for this activity with an extra restriction in our WHERE-clause. The date and time of a new change (record) is identified in the CDPOS table with respectively fields **UPDATE** and **UTIME**. For example, to retrieve occurrences of the activity Change Purchase Requisition, where t_1 is 23.02.2011 10:39:47, we can perform the following query:

```
SELECT * FROM CDPOS, CDHDR
WHERE TABNAME='EBAN' AND FNAME<>'LOEKZ' AND CDPOS.CHANGENR = CDHDR.CHANGENR
AND ((CDHDR.UPDATE = '2011-02-23' AND CDHDR.UTIME > '10:39:47') OR CDHDR.UPDATE
> '2011-02-23')
```

Whereas, the *original* was:

```
SELECT * FROM CDPOS, CDHDR
WHERE TABNAME='EBAN' AND FNAME<>'LOEKZ' AND CDPOS.CHANGENR = CDHDR.CHANGENR
```

We do not have to set an upper limit for the date and time in this query (i.e. t_2) because we always update according to the current state of the database. When a real-time connection between the prototype and the SAP database would be present, it might be interesting to update **to** a certain timestamp as well. Furthermore, additional attributes that should be

retrieved from other tables are assumed to be present in our database due to implementation decision **D1** (Section 7.1.2). For example, a change to a purchase requisition can only occur if the purchase requisition is created earlier. This implies that information about this purchase requisition is available.

SPLIT

This class of activities deals with updates in exactly the same way as the **CHANGE** class does. The difference with the **CHANGE** class is the fact that the **TABKEY** field (in **CDPOS**) could not directly be linked to the case representation. To create a case for such a change we had to look up the case attributes in another table by means of the **TABKEY**. Again, we can assume that those case attributes in this other table are present, since without these attributes, and thus without the record, the change could have never been done in the first place. This idea is again guided by decision **D1**. So it suffices to add a constraint to our SQL query to only select changes that occurred after the event log extraction date: i.e. after t_1 .

SQL

The third class of activities requires a bit more care however. To detect these activity occurrences we do not make use of the timestamp idea. The reason for this is that some events can otherwise not be detected due to missing timestamp information of the actual change. To deal with this problem we introduce the notion of **extraction flags**. Extraction flags indicate if a record in a table is extracted before. This means that, if during a previous event log extraction an event is retrieved from this record, this record should not be considered in a subsequent extraction (the incremental update). To support this we have to add a boolean field to each table (except **CDHDR** and **CDPOS**) in our local database which represent the extraction flag.

As you might guess, these flags have to be set upon completion of a regular event log extraction process as well. Initially all extraction flags are set to false; the last step of the procedure presented in Section 5.4 now is to set all extraction flags to true in the tables that were consulted during the event log extraction (excluding **CDPOS** and **CDHDR**). Also if the record is not used we set the flag, this has no consequences since if it is not used, it implies that no event existed in this record. Since we are not aware of activities where, if we set an extraction flag of a record to true, this record is later updated with new values that indicate another event, this approach is viable (Assumption **A1**, Section 7.1.1).

We also set the extraction flags to true once an update is finished, similar to a regular event log extraction. So, when we want activity occurrences after timestamp t_1 , we can extend our **WHERE**-clause to filter on extraction flags that are **false**, because all activities before t_1 have an extraction flag of true, and after t_1 of false. Retrieving all Creations of Purchase Requisitions in an updated database can be done as follows:

```
SELECT * FROM EBAN WHERE EXTRACTED <> true
```

This approach could also be used in the other two activity classes, however, due to the sheer size of these change tables, setting extraction flags in **CDPOS** and **CDHDR** would require too much time, and a timestamp approach gives the same result.

Addition to Process Repository

These new SQL queries are constructed with the help of a new file that is added to our process repository: `<ProcessName>activitiesToTablesInc.csv`. The file is very similar to the `<ProcessName>activitiesToTables.csv` file, the query classes (SQL, CHANGE and SPLIT) again denote how our prototype should construct and handle the queries. For the SQL class we have to change the WHERE-clause of the query in order to filter on extraction flags.

8.3 Technical Structure

The functionality our prototype provides and implements was presented in the previous sections and chapters, this section provides some more technical insights on our actual implementation in Section 8.3.1, and a class diagram that presents the architecture of our prototype in Section 8.3.2.

8.3.1 Implementation Details

Our prototype is written in the Java programming language, using Eclipse as our software development environment. A connection to the local PostgreSQL database is laid through a PostgreSQL JDBC driver. The prototype allows a user to connect to a different type of database. The **Driver** and **Connection** string, necessary to connect to a database from Java, can be specified through the GUI in our prototype. This is tested with SQL Server and proven to work. However, SQL Server's SQL implementation is slightly different than PostgreSQL's SQL implementation which made it necessary to modify the base SQL queries.

8.3.2 Class Diagram

A class diagram of our prototype is depicted in Figure 8.3. The class diagram is based on the OMG UML 2.0 specification¹ and contains the Java classes and interfaces of our prototype. The most important classes are included, some uninteresting classes are left out.

Each class is represented by an entity, dependencies and associations are indicated by the lines connecting them. A solid line with a normal arrowhead represents an **Association**. Associations between classes most often represent instance variables that hold references to other objects. We can see for example an association relation between *TabPanel* and *EventLog*, the direction of the arrow tells us that *TabPanel* holds a reference (0 or 1) to *EventLog* through instance variable *eventLog*. Solid lines with the crossed circles in the end signify **Nesting** on the other hand. A nesting relation shows that the source class is nested within the target class (at the encircled cross). The 'listener classes' *EventLogListener* and *TableCaseMappingListener* are for example nested in *TabPanel*. A dotted line indicates **Dependency**, a form of association. This means that one entity depends on the behavior of another entity because it uses it at some point of time (a class is a parameter or local variable of a method in another class). The arrowhead indicates asymmetric dependency, for example, the *CaseCalculator* class depends on the *TableCaseCalcutor* class.

¹<http://www.omg.org/uml/>

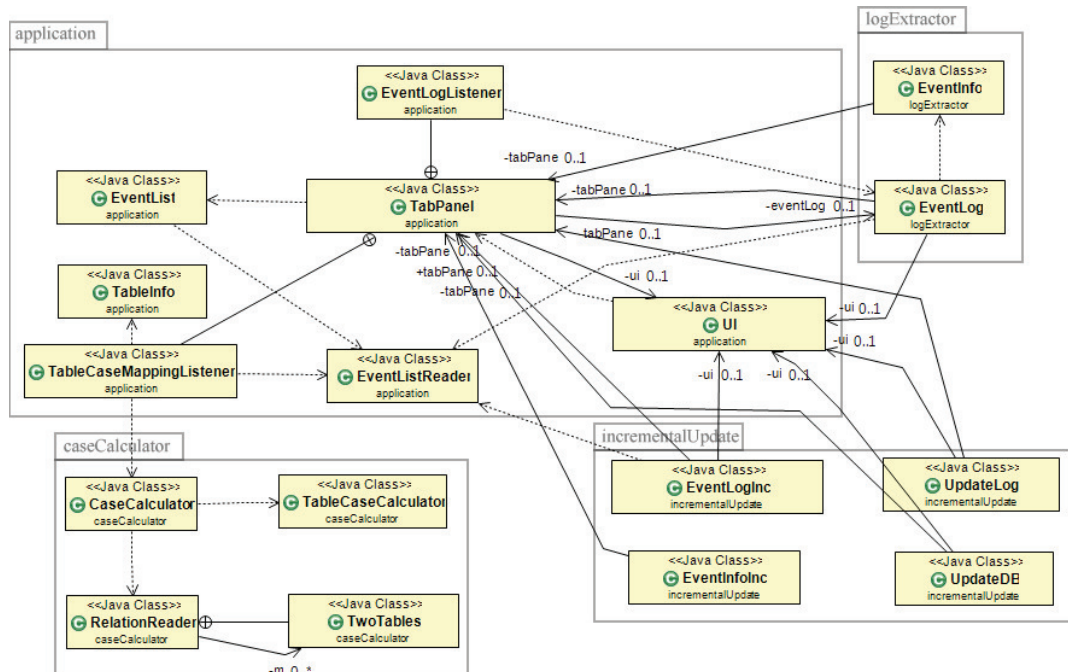


Figure 8.3: Class Diagram Prototype

We can identify four packages in which our classes reside. The main package, **application**, provides the user interface. The most important class here is *UI*, which builds up the entire graphical user interface and defines actions that are accompanied with buttons etc. From the user interface we can actually execute two important actions: (1) retrieving table-case mappings and (2) extracting the event log. Retrieving the table-case mappings is done through classes provided in the package **caseCalculator**. An important step here is to retrieve all foreign key relations from the accompanied CSV file, which is done by class *RelationReader*. Extracting the log is performed in the package **logExtractor**. The class *EventLog* implements the algorithm that is sketched in Section 5.4 (from step 2 onwards) and is responsible for the extraction of the event log, treating each activity as discussed in Section 8.1.1. It is supported by functionality provided in class *EventInfo* to connect to our target database and execute the SQL queries. The fourth package, **incrementalUpdate**, implements our incremental update procedure. The updating of the local database is done through class *UpdateDB*, which also provides the GUI for this step. The routine to update the event log is started in class *UpdateLog*, the actual algorithms and support to connect to the local database is found in classes *EventLogInc* and *EventInfoInc* respectively.

8.4 Graphical User Interface

We now present the graphical user interface of our prototype and show how to execute the most important steps in an event log extraction procedure. That is, from determining the possible table-case mappings to extracting the event log with a selected mapping. An example of a database and event log update is given at the end of this section.

8.4.1 Selecting Activities

An overview of the graphical user interface of our prototype is given in Figure 8.4.

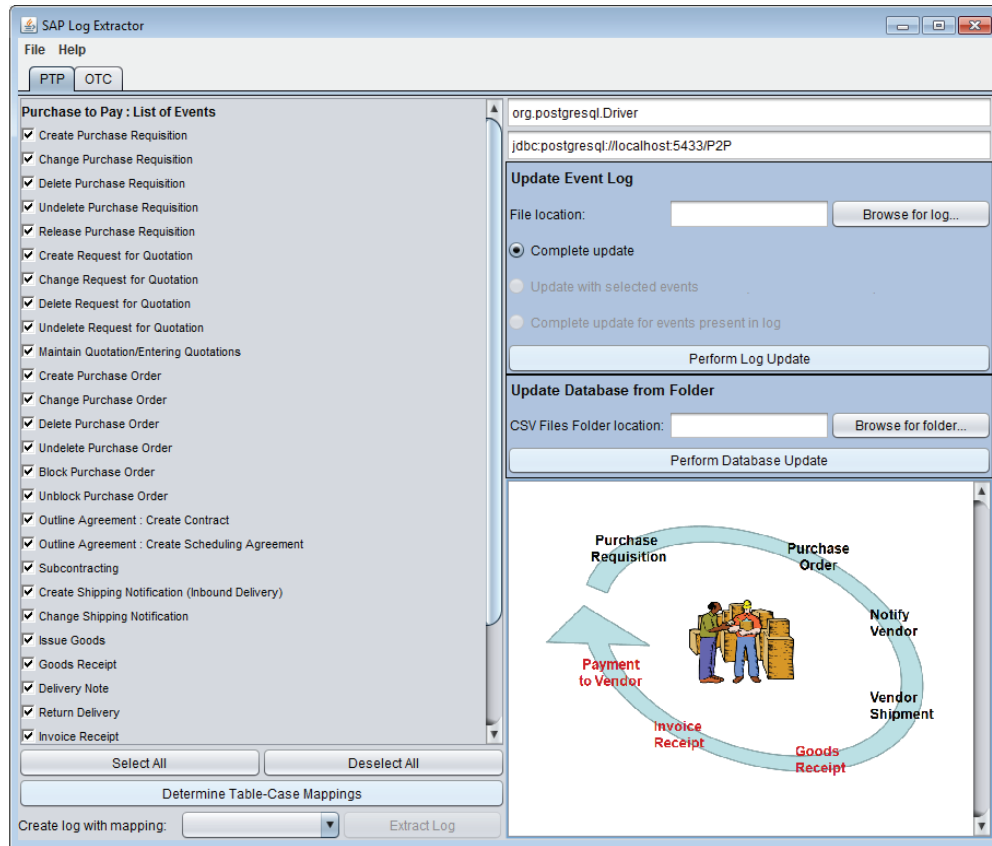


Figure 8.4: Overview GUI Prototype

Each SAP process that can be mined by the prototype has a separate tab, in the screenshot the PTP process tab is opened. These tabs are built by using information contained in the process repositories. The left side of the tab panel shows a list of activities related to the PTP process. The user can select the ones he/she wants to include in the event log extraction, or select, deselect them all. The driver and connection string needed to connect to the local copy of our database can be found in the top right corner. It is possible to change these settings such that another (type of) database is used. The two panels below, *Update Event Log* and *Update Database from Folder* deal with the incremental updating of previously extracted event logs. The panel in the bottom right corner (picture in Figure 8.4) is used to display messages to the user and can be seen as some sort of console.

8.4.2 Computing Table-Case Mappings

Once the activities have been selected, the user can push the *Determine Table-Case Mappings* button to calculate possible table-case mappings. If there exists a common case representation

between all activities selected, the console on the bottom right first outputs all tables involved with these activities, followed with a list of table-case mappings (the procedure to compute these is given in Section 6.1.3). Figure 8.5 shows us the results when table-case mappings have been determined for all activities in the PTP process.

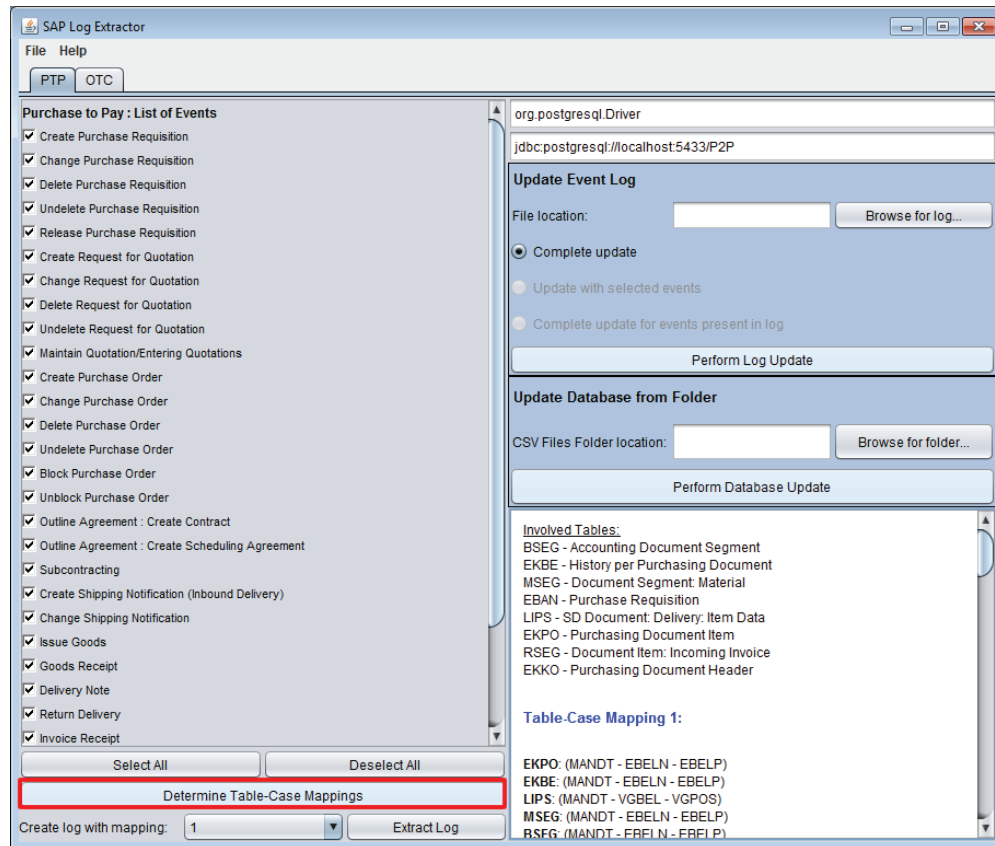


Figure 8.5: Determining Table-Case Mappings

8.4.3 Extracting the Event Log

The next step is to select one of these table-case mappings. This mapping identifies the case throughout the process, and specifically indicates the fields that represent this case per table. From Figure 8.6 we can observe that there are eight possible table-case mappings; another interesting fact is that table-case mapping 2 and 3 have a different number of fields; this means that cases are identified on different levels.

Once a table-case mapping has been chosen from the drop-down box, the user can push the *Extract Log* button to start extracting the log with the preferred mapping. Figure 8.7 shows us an event log extraction in progress. The user is made aware of the progress of the extraction with a progress bar, showing the activity currently being extracted and the percentage of completeness.

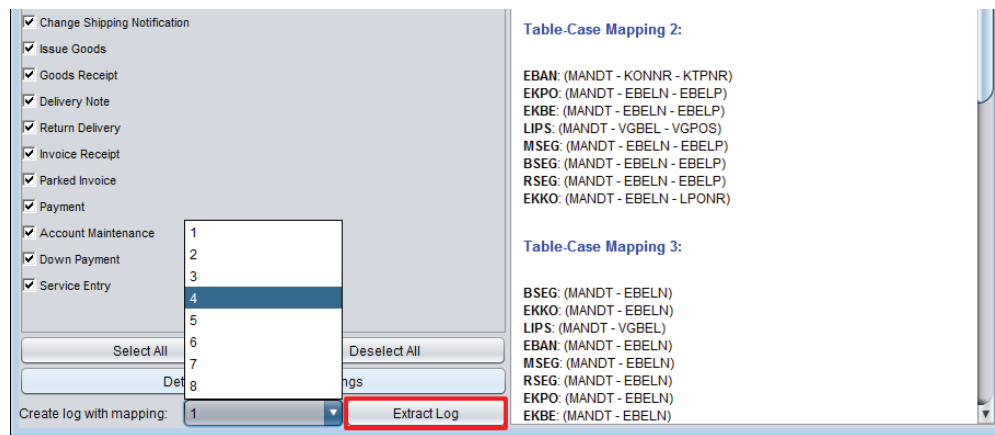


Figure 8.6: Choosing a Table-Case Mapping

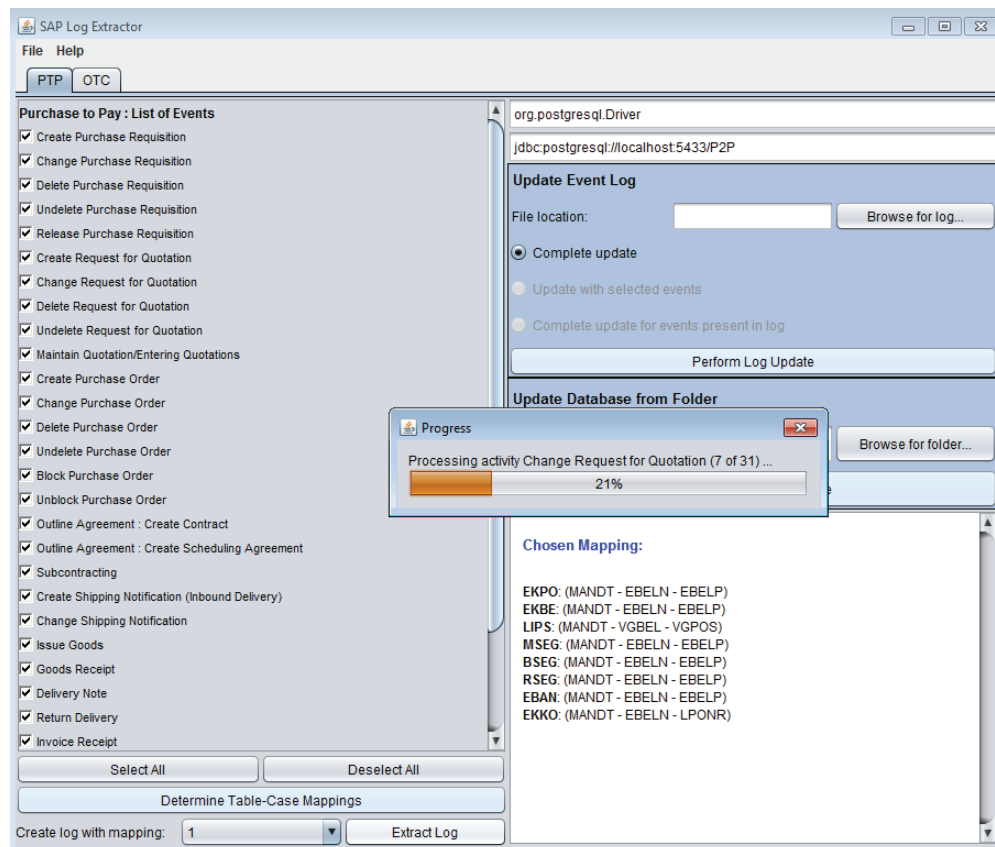


Figure 8.7: Extracting the Event Log

8.4.4 Extraction Results

When the extraction is complete, the user is informed about the elapsed time of the extraction (see Figure 8.8) and the resulting event log in CSV format is written to the prototype's root folder. The file name of an extracted event log is as follows: '<ProcessName> <timestamp>.csv'. For an event log extracted for the Purchase To Pay process on 26-01-2011 at 10.56.50 the filename is '**PTP 26-01-2011 10.56.50.csv**'. Listing 8.9 shows us an excerpt from an extracted event log. We can observe several events for activities *Invoice Receipt* and *Payment*, including some key attributes (case, executor, timestamp) and additional attributes. The first line of the event log indicates the meaning of each column in such a row (i.e. for one event). For this file that line would read: <Case ID;Activity;Key 1;Key 2;Key 3;Resource;Timestamp;...;Amount in Local Currency;Amount in document currency;...>.

The case studies presented in Chapter 9 clarify event log extraction through our prototype further and shows an analysis of these extracted event logs with Futura Reflect.

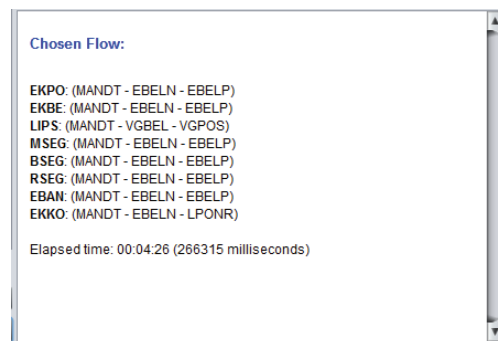


Figure 8.8: Event Log Extraction Complete

```

74538 837;Invoice Receipt;800;4500007715;10;HAMED;1999-11-15 5:02:42;;;;;3.620,00;;;;;
74539 7812;Invoice Receipt;800;3000000122;5;GRAUENHORST;2001-05-07 16:28:38;;;;;64,99;;;;;
74540 21134;Invoice Receipt;800;4500012559;10;OLBERT;2001-12-18 14:01:21;;;;;25.500,00;;;;;
74541 19404;Invoice Receipt;800;4500013080;10;HAMED;2002-03-18 5:02:47;;;;;13.515,00;;;;;
74542 10365;Invoice Receipt;800;4500014723;40;I802358;2002-12-05 5:11:20;;;;;27.785,60;;;;;
74543 3897;Invoice Receipt;800;4500015198;40;MAASSBERG;2003-04-04 4:03:25;;;;;38.712,60;;;;;
74544 26972;Invoice Receipt;800;4500015305;40;MAASSBERG;2003-06-05 4:01:46;;;;;40.446,00;;;;;
74545 6275;Payment;800;3000000122;1;OLBERT;2001-01-05 17:17:29;;;;;1.152.669,76;;;;;
74546 11852;Payment;800;4500007976;20;OLBERT;2000-02-21 19:12:27;;;;;18.000,00;;;;;
74547 6287;Payment;800;3000000168;4;OLBERT;2001-01-05 16:58:44;;;;;1.152.669,76;;;;;
74548 7902;Payment;800;414-0200;80;OLBERT;2001-01-05 17:52:12;;;;;796.700,00;;;;;
74549 27694;Payment;800;4500004582;20;D023346;1998-03-03 10:59:56;;;;;2.004.353,40;;;;;
74550 594;Payment;800;4500001432;50;D023346;1999-08-23 5:51:12;;;;;344.364,50;;;;;

```

Listing 8.9: Excerpt of an Event Log Produced by the Prototype

8.4.5 Updating the Database

The local database can be updated with a collection of CSV files (one for each table) that contain the new data. In Figure 8.9, the panel that allows for doing this is delineated. By pressing the button ‘*Browse for Folder...*’, a folder can first be selected that contains these CSV files, subsequently, the button *Perform Database Update* starts the actual update procedure. Each table is brought up to date with the algorithm presented in Section 7.2.1.

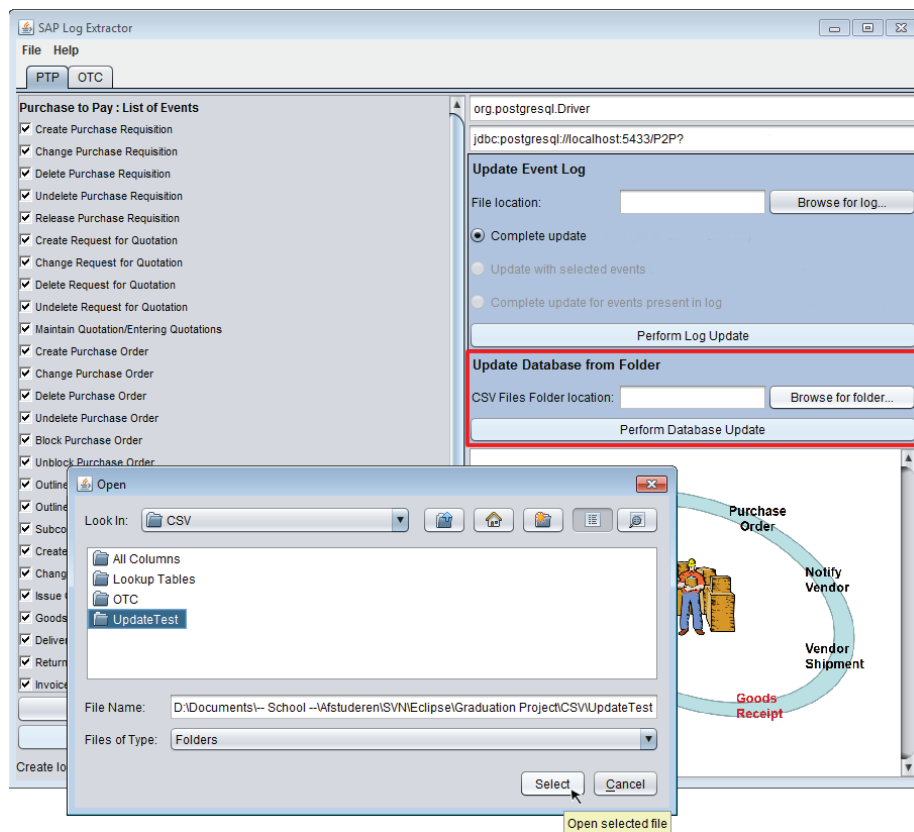


Figure 8.9: Selecting the Database Update Folder

8.4.6 Updating the Event Log

In order to actually update the event log, the file location of the event log you want to update needs to be specified first. Currently, the only update option present is to update the event log according to the current state of the database. The selecting of activities to include in the event log update can still be performed. Figure 8.10 shows the *Update Event Log* panel delineated and an event log update in progress. The button ‘*Browse for log...*’ allows for specifying the location of the event log file; the update is started by pressing the button *Perform Log Update*. This procedure follows the algorithm described in Section 7.2.3.

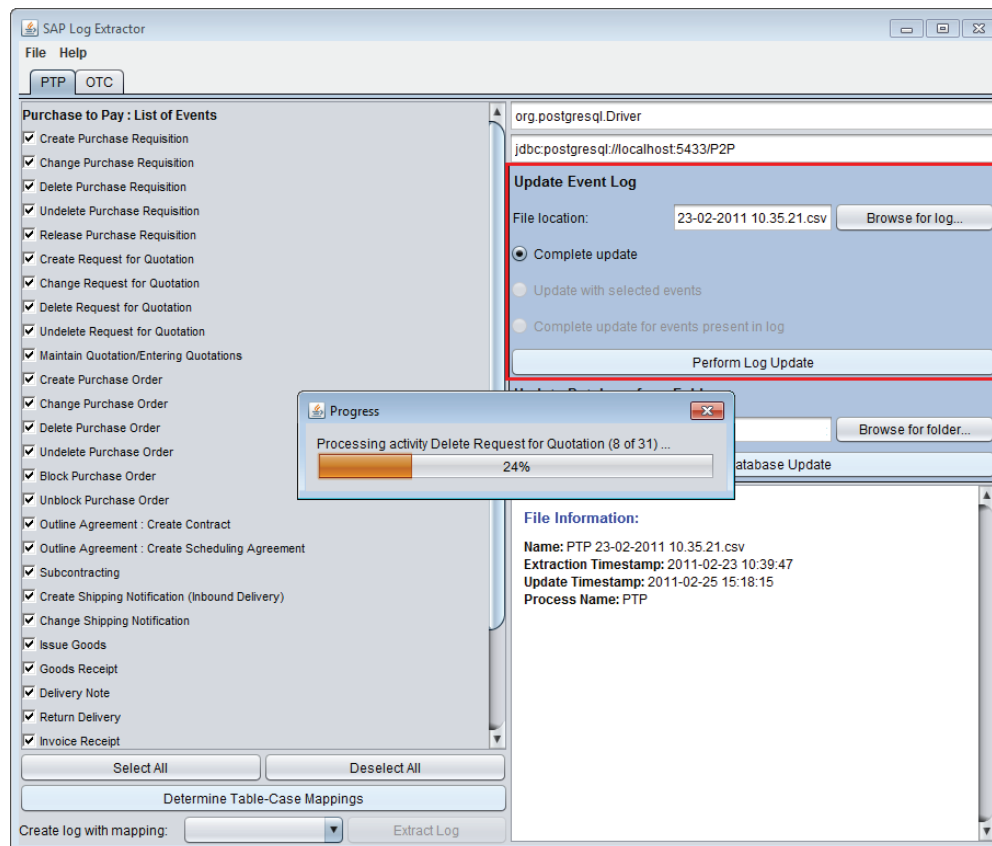


Figure 8.10: An Event Log Update in Progress

As we can observe from the figure we are currently processing the activity *Delete Request for Quotation*. The event log we are updating is called *PTP 23-02-2011 10.35.21*, which is extracted on 2011-02-23 at 10:39:47 and was last updated on 2011-02-25 at 15:18:15.

Results

When the updating of the event log is complete, all newly extracted events are appended to the event log. This file can then be analyzed further with Futura Reflect in order to detect important changes in the process model. The time necessary to actually extract and write

the events to the log file is linearly related to the number of events. So typically an event log update would require less time than an entire log extraction, since updates often concern less events.

8.5 Incremental Update Improvements

There are several improvements or additions we can think of for our prototype regarding the updating of event logs. The current functionality of our prototype suffices to update the local database and perform an event log update based on this updated database. However, since this is the first attempt in incrementally updating event logs (for SAP), improvements can always be proposed. The most important ones are as follows:

1. Creating a direct coupling between the prototype and the SAP database. This would allow for a much quicker event log update since then we do not have to update the local database. Even more, event logs could possibly be updated continuously which can then again lead to *continuous process monitoring*. It is possible to execute SQL queries on the SAP database; however, the setting of extraction flags in the actual SAP database is not possible. We have to think of other methods to deal with this; e.g. locally storing which records of a table were already used in a previous extraction.
2. Extend the event log update options with the possibility to (in addition to a complete update):
 - update an event log with events that occurred between certain timestamps.
 - only extract the activities that reside in the current event log.
3. If multiple events (that occur on different timestamps) can be retrieved from exactly the same database record, review the extraction flag/timestamp approach. Possibly, extraction flags could be set per field of the table.
4. Setting extraction flags during an initial event log extraction is time consuming when dealing with large tables; find other mechanisms to do this.
5. Updating an event log results in changes in the extraction fields of some tables in our local database. This means that the update of another event log uses this same version of the database (where possibly some extraction flags were already set). Event logs and the database are thus coupled at the moment. For completely extracting two event logs, using different table-case mappings, this does not make a difference. We do have consequences when we want to update these two event logs with the same data; for the activities that are extracted from the change table this does not make a difference, the activities which we retrieve by using the extraction flags would however be missed in a second extraction.

Most improvements concern adding functionality to our application prototype. Only improvement number three would be a *conceptual extension* of our prototype. This improvement would become interesting if a business process is found where our timestamp/extraction flag approach would not work.

8.6 Conclusion

In this chapter we presented our prototype and explained how it implements our event log extraction procedure from Chapter 5, using the table-case mapping approach from Chapter 6. It explained the configuration files that need to be created and set up for each process in order to perform an event log extraction for that process, and indicated the importance of having a repository for this. Our incremental event log update procedure from Chapter 7 was embedded into our prototype and the changes that have to be made to the process repository to support this were discussed. Furthermore, we presented the technical details about the structure of the prototype as well as a graphical introduction to the user interface. We concluded by critically discussing some improvements that can be made to our implementation of the incremental update procedure.

Comparing our prototype to Buijs' XES Mapper [4], retrieving event occurrences by setting up SQL queries is of course a similar approach, but the analogies only go as far as that SQL is a standard way to retrieve information from a database. In this project the queries are first of all stored in a repository, secondly the queries are made such that they support the selection of different cases (table-case mappings). Furthermore, selection of important attributes (e.g. timestamps) and additional attributes (e.g. price and vendor information) is not included in these base SQL queries, but are added as necessary and as configured in our prototype, giving each event log extractor its desired level of detail and allows having multiple views on the process.

An event log extraction with our prototype encompasses two things: (1) the configuration of our prototype through the process repository CSV files, and (2) the actual event log extraction using the GUI the prototype offers. Additionally we have proven that SAP allows for incremental updating of event logs extracted for the PTP and OTC process. We could generalize this as a characteristic of SAP, *updating of event logs extracted from SAP is feasible*. There were however some improvements that could be identified; these mostly concern the prototype implementation in general, as well as some ideas to give more options to the person performing an event log update. Speed issues were caused by having to update a local database and setting extraction flags, this deserves some more investigation in the future however. A general improvement we could make to the prototype is to further automate the data extraction procedure. Open Source tools like Talend show that this is feasible, and even allow a connection to a local database.

Chapter 9

Case Studies

We have implemented two processes in our prototype as a proof of concept: the **Purchase to Pay** process and the **Order to Cash** process. During construction of the prototype we continuously and extensively tested the prototype using (parts of) the PTP process. This process was addressed several times throughout this thesis and is discussed further in Section 9.1. A process repository for the OTC process was created upon completion of the prototype. Learning to execute the OTC process in SAP and configuring this repository took about *one week*. A case study on the OTC process is presented in Section 9.2. We conclude this chapter in Section 9.3 by discussing the mining results and the applicability of our prototype. In both case studies we specifically focus on the event log extraction with our prototype, as well as the analysis with Reflect. For setting up SQL queries and other preparation activities we refer to Chapter 5 and 8. We thus assume that the process repositories have been created.

9.1 Purchase To Pay

The PTP process was introduced in our preliminaries in Section 2.1.3. It focuses on *procurement of trading goods* and is considered as one of the most well-known and implemented processes in SAP. In the following sections we first outline the activities in this process (Section 9.1.1) and analyze the tables that are used (Section 9.1.2). Subsequently we extract events log for the **entire** PTP process, using two different table-case mappings. These event logs are analyzed with Futura Reflect in Section 9.1.3 and 9.1.4. In Section 9.1.5 we compare both process mining results and discuss the influence of table-case mappings on the models. A small section is dedicated to showing our prototype work on a **subset of activities** (Section 9.1.6), which requires the use of a totally different case representation. Section 9.1.7 exemplifies how an update is actually performed through our prototype.

9.1.1 Activities

With the method described in Section 5.3.1 we can determine all important activities in the PTP process. There are **31** activities; these are listed in Table 9.1. As was addressed before, much more activities could be identified in this process if we would ‘use’ the change tables more. Several *change table activities* are now captured under one ‘Change activity’, like changing the order amount and delivery date. Deletion and blocking of purchase orders are the only ‘Change activities’ that are split up from this; much more change activities on

purchase orders could be retrieved in a similar way or even *automatically discovered*.

Table 9.1: Purchase to Pay Activities

Create Purchase Requisition	Change Purchase Requisition
Delete Purchase Requisition	Undelete Purchase Requisition
Release Purchase Requisition	Create Request for Quotation
Change Request for Quotation	Delete Request for Quotation
Undelete Request for Quotation	Maintain Quotation
Create Purchase Order	Change Purchase Order
Delete Purchase Order	Undelete Purchase Order
Block Purchase Order	Unblock Purchase Order
Outline Agreement : Create Contract	Create Scheduling Agreement
Subcontracting	Create Shipping Notification (Inbound)
Change Shipping Notification	Issue Goods
Goods Receipt	Delivery Note
Return Delivery	Invoice Receipt
Parked Invoice	Payment
Account Maintenance	Down Payment
Service Entry	

9.1.2 Table Characteristics

Before we start extracting our event log we present some information about the number of records in each table that we use. This gives an idea about the scale of the PTP process; Table 9.2 presents this overview.

Table 9.2: Number of Records in Purchase to Pay Tables

Table	# Records	Table	# Records
BKPF	257,753	BSEG	943,636
CDHDR	567,797	CDPOS	3,644,087
EBAN	3,046	EKBE	52,104
EKET	27,839	EKKO	13,855
EKPO	28,027	LIKP	11,726
LIPS	20,379	MKPF	65,278
MSEG	115,737	RBKP	5,507
RSEG	14,543		

9.1.3 Purchase Order Line Item Level

In this section we perform an event log extraction for the complete PTP process. The introduction to the graphical user interface in Section 8.4 showed us a first glimpse on how to start an event log extraction for the PTP process. We follow these same steps and select all activities within the PTP process. From the computed table-case mappings we use the following table-case mapping to extract our event log:

EKPO: (MANDT - EBELN - EBELP)
 EKBE: (MANDT - EBELN - EBELP)
 LIPS: (MANDT - VGBEL - VGPOS)
 MSEG: (MANDT - EBELN - EBELP)
 BSEG: (MANDT - EBELN - EBELP)

RSEG: (MANDT - EBELN - EBELP)
 EBAN: (MANDT - EBELN - EBELP)
 EKKO: (MANDT - EBELN - LPONR)

The semantics of the three fields implies that we chose a table-case mapping for the PTP process on a *purchase order line item level*. Extracting the event log with our prototype results in a CSV event log file with a size of 19,9 MB. This file can then be imported in Reflect by importing it as a new dataset. The event log contains **230,580 events**, spread over **33,248 cases**. There are **19** different types of **activities** extracted, Figure 9.1 gives the number of events per activity.

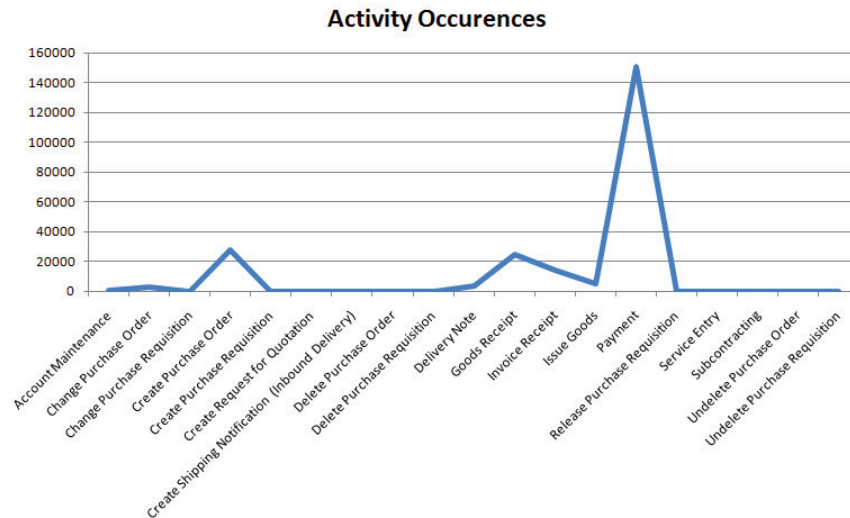


Figure 9.1: PTP Events per Activity

The timestamp the first event occurs is Nov 29, 1994 12:56:14, while the last event occurs on Dec 3, 2010 12:37:42 PM. The process model discovered by using the Genetic miner with a target completeness percentage of 90% is shown in Figure 9.2. The target percentage indicates how many cases a mined model should capture. The screenshot provides an overview of Reflect as well; the most common actions are listed in the left panel: *Overview*, *Mine*, *Explore*, *Animate* and *Charting*. The **Mine** functionality we used discovers the process model that best describes the behavior of the complete cases in the current dataset.

Another commonly performed task in Reflect concerns the *exploring* of a dataset. The **Explore** functionality discovers the process model that describes a certain percentage of cases (complete or not) in the dataset. Figure 9.3(a) shows us a process model that considers **90% of the cases**. In this discovered model dark purple portrays the most frequent path followed by the majority of the cases. The colors will fade as the paths become less frequent. Compared to the Mine functionality, the models mined by using the Explore functionality do not support parallel constructs, are based on complete as well as incomplete cases, are simpler than the ones discovered using the Mine functionality because ‘Explore’ models do not support parallel constructs, and are based on complete as well as incomplete cases. The model is created from 29924 cases (90%) and fits 30298 cases (91%) out of 33248 cases. It is possible

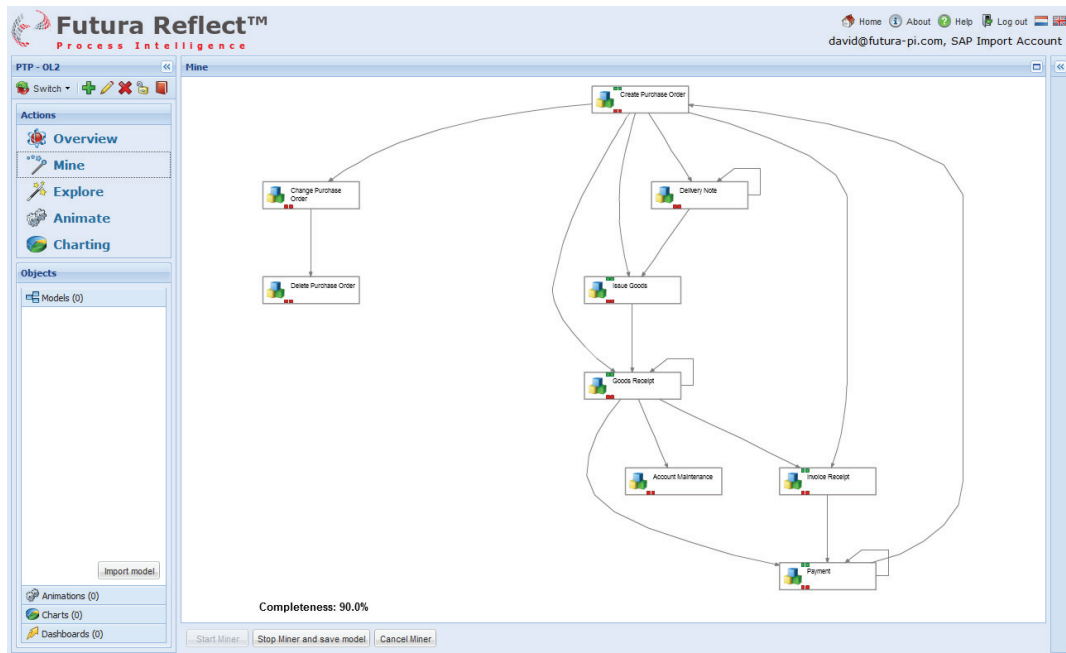
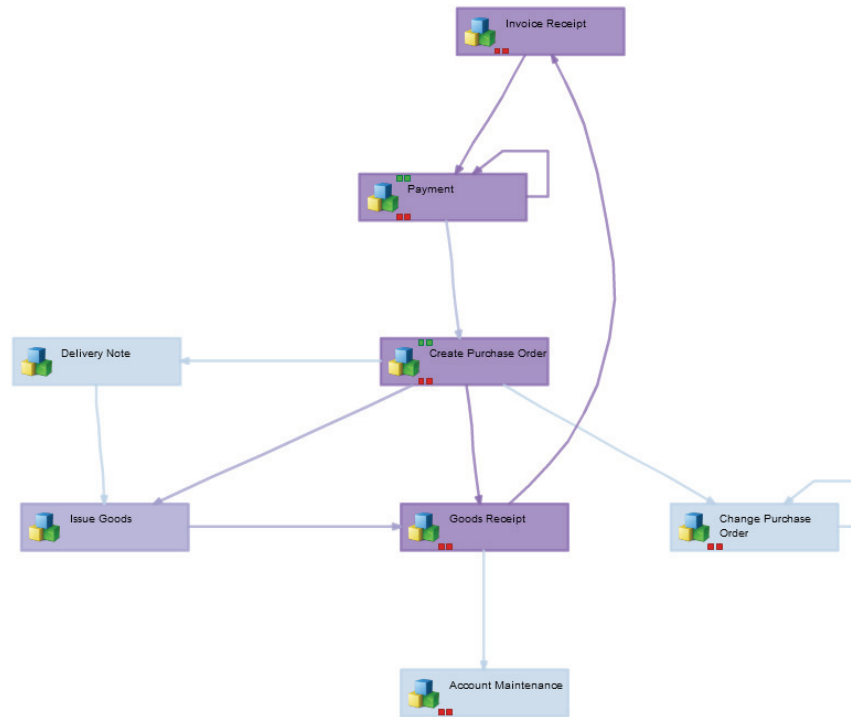


Figure 9.2: Genetic Miner Model

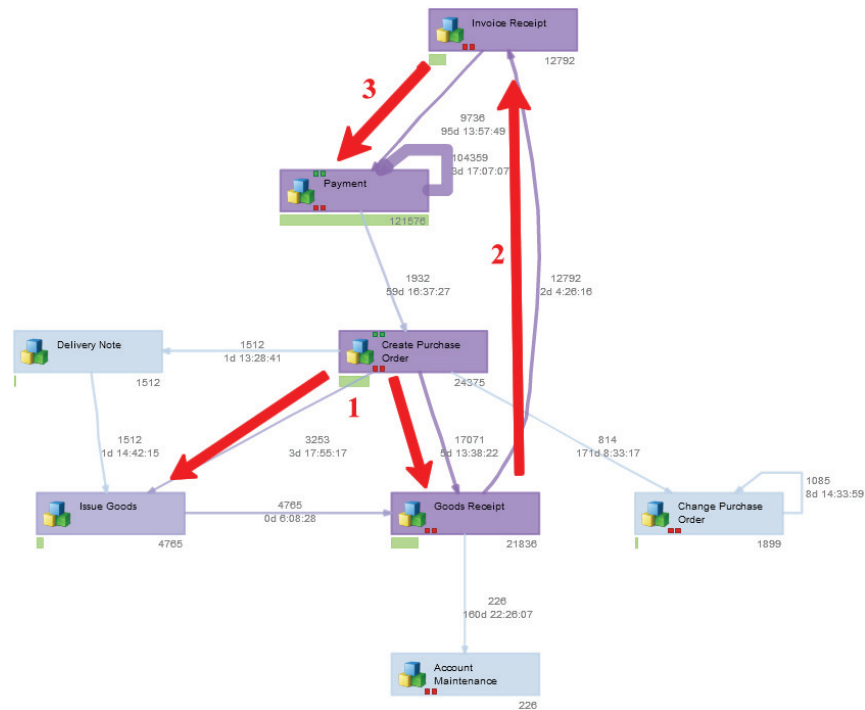
to apply performance analysis on the constructed model as well, Figure 9.3(b) depicts that same process model with the performance metrics projected on it; the red numbered arrows were added to indicate the main flow of events.

Figure 9.3 thus presents us a first view on the basic flow of the PTP process, mined on Purchase Order Line Item level. The basic sequence of actions is: *Create Purchase Order*, *Issue Goods*, *Goods Receipt*, *Invoice Receipt* and *Payment*. Furthermore we can observe from the performance metrics in Figure 9.3(b) that payment events occur more frequently than other events. This is due to the characteristics of the IDES database, and the (probably) auto-generated data in the databases. In the BSEG table we for example find multiple payments for an invoice that belongs to a Purchase Order Line Item, spread over multiple terms, sometimes recurring each year. This is also indicated by the self-loop for the activity Payment, which indicate that (at least) two subsequent payment actions for the same purchase order line item are not intervened by another type of event.

A more complete look on the process is acquired by including more cases. Figure 9.4(a) presents a model that is created from 32916 cases (99%) and fits 32950 cases (99%) out of 33248 cases. Even this model is pretty structured and has a clear basic flow. Some things to observe: there are only 53 Purchase Order Line Items created based on a Purchase Requisition, and 28 Purchase Order Line Items were immediately deleted after creation. If you would include all events in the process model (a model that fits 100% of the cases) you unavoidably receive a ‘spaghetti’ model. All possible sequences of paths are depicted in that model (Figure 9.4(b)).



(a) Without Performance Metrics



(b) With Performance Metrics

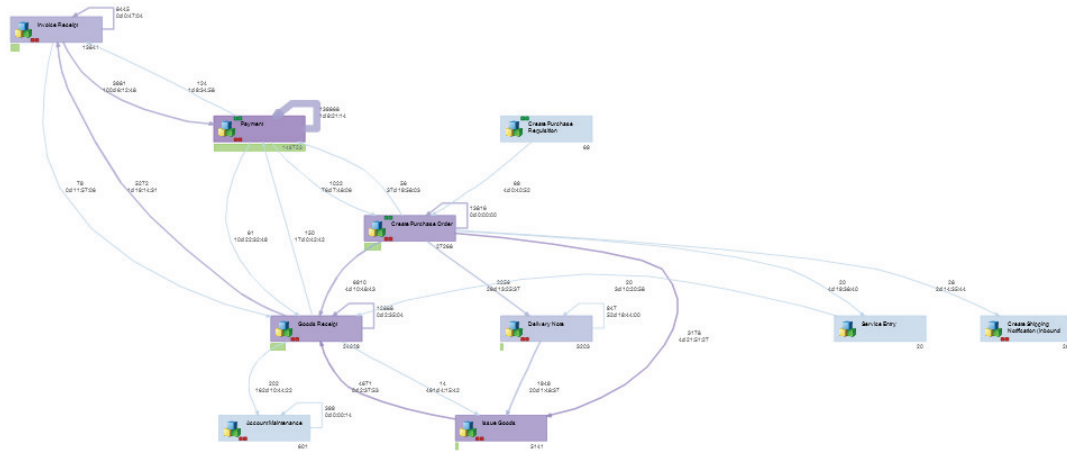
Figure 9.3: Exploring the PTP process on 90%



In this section we analyze the PTP process on a higher level, that is, we only look at Purchasing Documents and do not make a distinction between line items in that purchasing document. The case is thus the Purchasing Document, in our prototype we use the table-case mapping computed below to extract the event log, considering all activities in the PTP process.

The extracted event log has a size of 18,8 MB, contains **227,037** events in **18,280** cases spread over *only* 13 activities this time. The activities we miss are activities that should be retrieved from the change tables. This is due to the fact that our prototype could not link the TABKEY to different table-case mappings at the moment. In Figure 9.5 we can find three models that were created with Reflect. The models show a lot of similarity with the process models mined in Section 9.1.3, where we maintained a purchase order line item view. There are however important distinctions to be made, these will be discussed in the next section.





(c) Exploring 99% of the cases

Figure 9.5: Exploring and Mining the PTP process On PO Document Level

9.1.5 Comparison

As is mentioned throughout this thesis, the chosen table-case mapping influences the characteristics of the event log and view on the discovered process model. In Section 6.2 we introduced the notion of *convergence* and *divergence*, we now discuss how this relates to our examples.

First of all we take a look at the *average number of events per case*. This can be calculated by dividing the number of events by the number of cases. To correctly compute this, we have to consider the exact same activities in both event logs. In this case we only look at the 13 activities that were logged in the Purchasing Document level event log (PD event log). The Purchase Order Line Item level (POLI event log) for these 13 activities has 227037 events spread over 33248 cases. Thus, the average number of events per case for the POLI event log is **6.83**, while for the PD event log this is **12.42**. There are almost twice the amount of events per case for the PD event log as for the POLI event log. By exploring the two event logs in the previous sections we also observed that the number of self-loops is much bigger with the PD event log than the POLI event log. We can analyze it further if we look at the distribution of the number of events per case. Figure 9.6 presents us two graphs that depict these distributions.

While having less types of activities in the PD event log, the average number of events per case is still much more than the POLI event log. In both graphs we observe that the maximum number of events in a case is (much) larger than the number of activities, this implicates that some activities have multiple occurrences in a case. If we recall the definition of divergence in Section 6.2.1: **the same activity being performed several times for the same process instance (case)**, we identify **divergence** in both event logs. More specifically: the amount of divergence that occurs is more or less twice as high when mining on a purchasing document level than on a more detailed purchase order line item level.

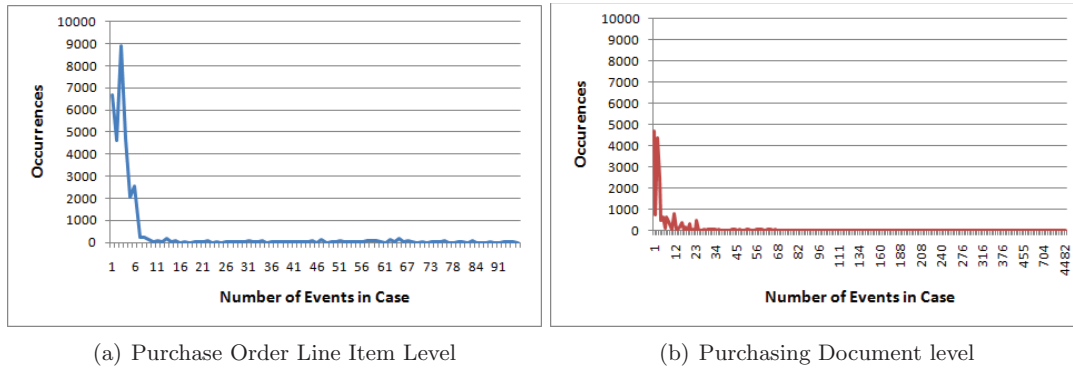


Figure 9.6: Number of Events per Case

Furthermore we can notice the existence of a few outliers in Figure 9.6(b): some cases contain a huge amount of events (e.g. 1302, 2002, 4482, 5548). These only occur once and concern Purchase Orders that contain many line items (e.g. 54 line items for order 4500010203), which are partially paid for as well. At PD level we do not distinguish between these payments which leads to grouping them in the same case. The difference between both graphs can be analyzed further, however, the idea is clear. In general, for our IDES SAP database, containing real-life test data, **the amount of divergence can be halved by choosing a different table-case mapping.**

Convergence, the same activity being performed in several different process instances, is a bit more difficult to detect. To do this we have to extract event logs where we include additional attributes that are able to uniquely identify such an activity. We illustrate this by extracting event logs and focusing on payments. To identify payments in an event log we need the attributes MANDT (Client), GJAHR (Year) and BELNR (Accounting Document) to be logged with payment events. We can then group cases that belong to the same accounting document, and set out how many cases belong to each accounting document. Of course cases can refer to multiple accounting documents at the same time as well (i.e. divergence), but that is not of our concern at the moment. The next step is to make a distribution of *how many cases on average belong to the same payment activities* (i.e. accounting documents). Table 9.3 illustrates this for the PD and POLI event log, it only shows the occurrences of payment activities that occur in up to 15 different cases. Payment activities that are being performed in more than 15 different process instances (cases) are not considered because their occurrence is (close to) zero.

The numbers are very alike in the table and it is hard to deduce something from it. We can make two observations however; (1) most payment activities only target one case (3985 out of 4646) and (2) the number of cases that refer to the same payment activity is more or less the same for the PD and POLI event log. We can however conclude and confirm that **SAP exhibits convergence of data.** We could look in detail and analyze the occurrences in both PD and POLI event logs; the same payment activities that occur in few

Table 9.3: Number of Cases per Accounting Document

Purchasing Document		Purchase Order Line Item		Difference
# Cases Payment Activity Considers	Occurrences	# Cases Payment Activity Considers	Occurrences	
1	3985	1	3860	125
2	189	2	105	84
3	93	3	63	30
4	91	4	67	24
5	30	5	22	8
6	53	6	53	0
7	18	7	35	-17
8	16	8	60	-44
9	10	9	31	-21
10	17	10	26	-9
11	7	11	20	-13
12	19	12	28	-9
13	4	13	4	0
14	12	14	18	-6
15	7	15	6	1

(1-5) process instances are more often detected at the Purchasing Document level, whereas the same payment activities that occur in more (7-14) process instances are more common at the Purchase Order Line Item level. A reason for this is unclear. For a higher number of process instances (15+), this difference however is negligible. In the example it is clear that the table-case mapping that is chosen influences the amount of convergence that will occur; however, this influence is so small that it is difficult to make a general conclusion on this.

9.1.6 Purchase Requisition Level

As mentioned in this thesis, the selected activities and table-case mapping determines the view on a process. In the PTP process we can for example, instead of looking at the entire PTP process, focus on **Purchase Requisitions**. To do this we only have to select activities that deal with Purchase Requisitions. Based on these activities, table-case mappings can then be computed. Due to the fact that purchase requisition activities are only related to table EBAN, the algorithm from Section 6.1.3 (that is implemented in our prototype) returns all foreign keys for table EBAN. With this, the prototype computes a total of **41** table-case mappings. It is up to the user to select a mapping; however, few table case-mappings actually link on purchase requisition numbers. Because we query the Change tables for some purchase requisition activities as well, and automatically link those activity occurrences by the TABKEY field in CDPOS to the purchase requisition number (primary key), we have to select a table-case mapping that is able to make this link. Figure 9.7 exemplifies how all this is set up in our prototype; for the event log extraction we use table-case mapping 40.

In less than 5 seconds we retrieve an event log that contains the five selected activities, listing 5782 events spread over 3046 cases. The first event occurrence is at Jun 24, 1992 12:00:00 AM, while the last event occurs at Oct 28, 2010 3:03:38 PM. Table 9.4 lists the event

frequencies per activity and Figure 9.8 the mined model.

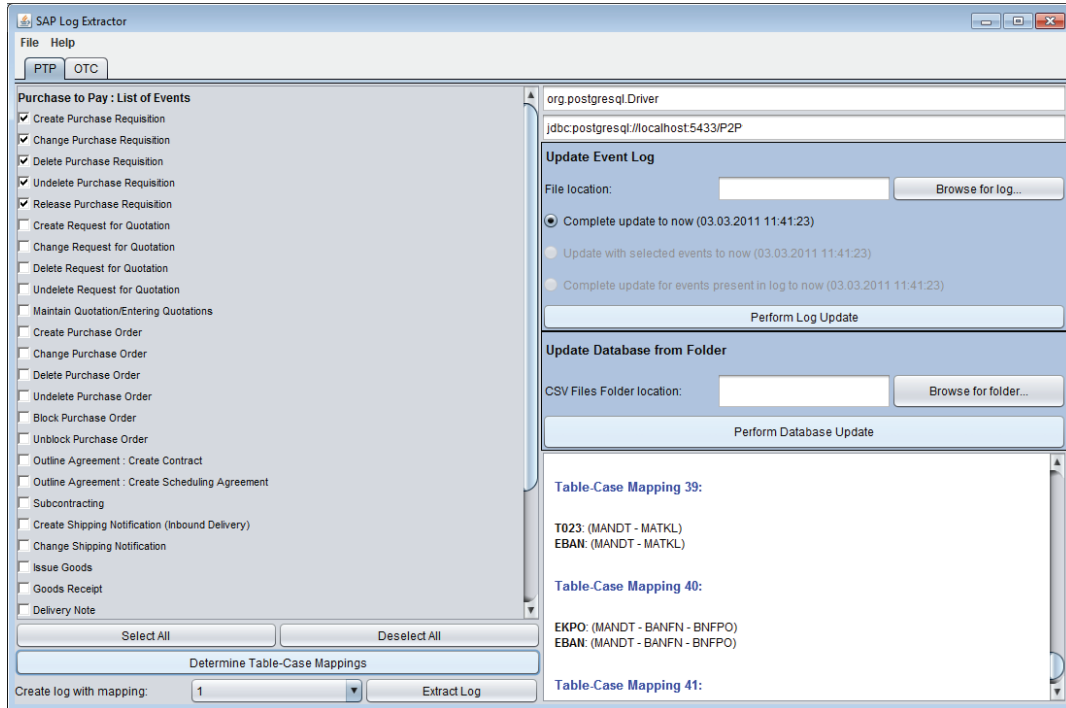


Figure 9.7: PTP Purchase Requisition Level

Table 9.4: Purchase to Pay Activities

Activity	Relative Occurrences
Create Purchase Requisition	8.02%
Change Purchase Requisition	52.68%
Delete Purchase Requisition	38.74%
Undelete Purchase Requisition	0.4%
Release Purchase Requisition	0.16%

Another table-case mapping that could be chosen is the one that takes the *Plant* as the case. In this scenario we then look at purchase requisitions from a *Plant* point of view, meaning that all purchase requisition items that are physically located in the same plant belong to the same case. When we extract such an event log (table-case mapping 7), we get an event log with 3046 events, spread over (just) 25 cases. This is of course due to the fact that plants contain multiple items, and many purchase requisition item need to be retrieved from the same plant. However, only one activity is recognized: *Create Purchase Requisition*. This is because the other activities are retrieved from the Change Tables and linking case attributes Client and Plant to the TABKEY in the change tables is not possible directly. We would have the look this up in the concerned base table.

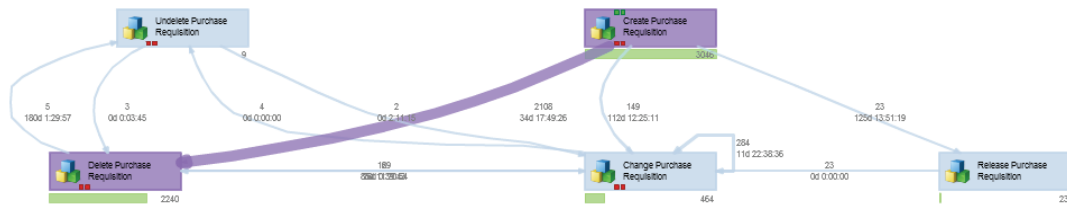


Figure 9.8: PTP Purchase Requisition Level: Mined Model

9.1.7 Incremental Update of an Event Log

In order to illustrate the updating of an event log, we first extract an event log on purchase order line item level like in Section 9.1.3. This event log, `PTP 16-01-2011 08.12.53.csv`, contains 230,580 events spread over 33,248 cases. The next step is to update our local database (that is up to date till 31-12-2010 23:59) with new data. This is data from events that occurred between 01-01-2011 00:00:00 and 17-03-2011 12:00:00. Table 9.5 presents the number of (new) records per table that we will try to insert in our local database.

Table 9.5: Number of Records in Update Data Tables

Table	# Records	Table	# Records
BKPF	7	BSEG	37
CDHDR	57,102	CDPOS	60,743
EBAN	19	EKBE	24
EKET	28	EKKO	27
EKPO	32	LIKP	25
LIPS	31	MKPF	34
MSEG	34	RBKP	32
RSEG	34		

The event log update is performed on a small scale; the change tables contain the most records since these contain other changes than just those for the PTP process. Due to the small size of the update it will be easier to verify whether our updated event log ‘equals’ an event log that is extracted from scratch with the updated database.

After we have performed the database update with the data above (following the procedure as explained in Section 8.4.5), it is time to update our event log. Here we again not show the actual steps that need to be performed within our prototype; these were already described in Section 8.4.6. Our updated event log (`PTP 16-01-2011 08.12.53.csv`) now contains 230668 events spread over 33281 cases. We thus have an addition of 33 cases and 88 events. The history log file is updated for this file as well; we now set the update timestamp to 17-03-2011 17:23:55 (the time of the update) such that future (incremental) updates use this timestamp instead of the original extraction timestamp.

Now the challenge is to check whether a new extraction on this updated database, with the same table-case mapping, results in the ‘same’ event log as we established by updating an event log. A normal extraction on the updated database gives an event log file `PTP 18-03-2011 10.18.19.csv`, it contains 230668 events spread over 33281 cases. These are

the same metrics as in our update event log file `PTP 16-01-2011 08.12.53.csv`. By looking up if each line in event log `PTP 18-03-2011 05.48.14.csv` occurs in the event log `PTP 16-01-2011 08.12.53.csv` and vice versa, we indeed have confirmation that both event logs contain the exact same events.

The size of the event logs slightly differs some kilobytes however. This is due to the fact that we include an integer case identifier with each event that identifies the case instance (on top of the case attributes). New data might lead to the fact that case instances have another case identifier than in the original event log; if a case that handles a lot of events is appointed a large integer, the file size will thus also change.

9.2 Order To Cash

The Order to Cash process supports the process chain for a typical sales process with a customer. It is introduced in Section 2.1.3 and is another frequently used ‘process’ in SAP. We do not discuss this process as detailed as the PTP process; Section 9.2.1 first lists the activities we identified in this process, the size of the tables we use to mine this process is given in Section 9.2.2 and Section 9.2.3 presents an event log analysis of the OTC process on *Sales Order Item level*.

9.2.1 Activities

Table 9.6 contains all activities we acknowledge for the OTC process. This is a total of **27** activities; detailed change activities are again not considered and captured under one ‘Change activity’.

Table 9.6: Order to Cash Activities

Create Sales Inquiry	Change Sales Inquiry
Create Sales Quotation	Change Sales Quotation
Create Standard Sales Order	Change Standard Sales Order
Post Goods Issue	Create Outbound Delivery (TO)
Create Shipment	Change Shipment
Confirm Delivery	Cancel Transfer Order
Packing	Goods Movement
Goods Movement (Documentation)	Billing the Sales Order
Change Billing Document	Invoice Cancelation
Intercompany Invoice	Pro Forma Invoice
Returns	Debit Memo
Debit Memo Request	Create Purchase Order
Create Contract	Credit Memo Request
Returns Delivery For Order	

9.2.2 Table Characteristics

Table 9.7 lists the number of records in each table that is used to extract the OTC process from SAP IDES. There are some overlapping tables with the PTP process (MSEG, MKPF, LIKP, LIPS), however different fields are queried.

Table 9.7: Number of Records in Order to Cash Tables

Table	# Records	Table	# Records
CDHDR	567,797	CDPOS	3,644,087
LTAP	16,669	LTAK	6,875
MSEG	115,737	MKPF	65,278
VBAP	14,571	VBAP	6,901
VBEP	19,361	VBFA	124,433
VBUK	49,549	VBUP	34,971
VBRK	30,860	VBRP	46,125
VTTK	47	VTTT	53
LIKP	11,726	LIPS	20,379

9.2.3 Sales Order Item Level

We now perform an event log extraction for the complete OTC process as presented in Section 9.2.1. If we use our prototype to retrieve table-case mappings, a total of 58 mappings are returned. The reason for this is that there are a lot of different relations between tables, the table-case mappings all exhibit small variants of each other. If we analyze these table-case mappings, we can observe as well that all these mappings contain **three fields**. When giving meaning to these mappings, all concern table-case mappings on a **sales-order item level**. The chosen table-case mapping, as well as the event log extraction in progress is found in Figure 9.9.

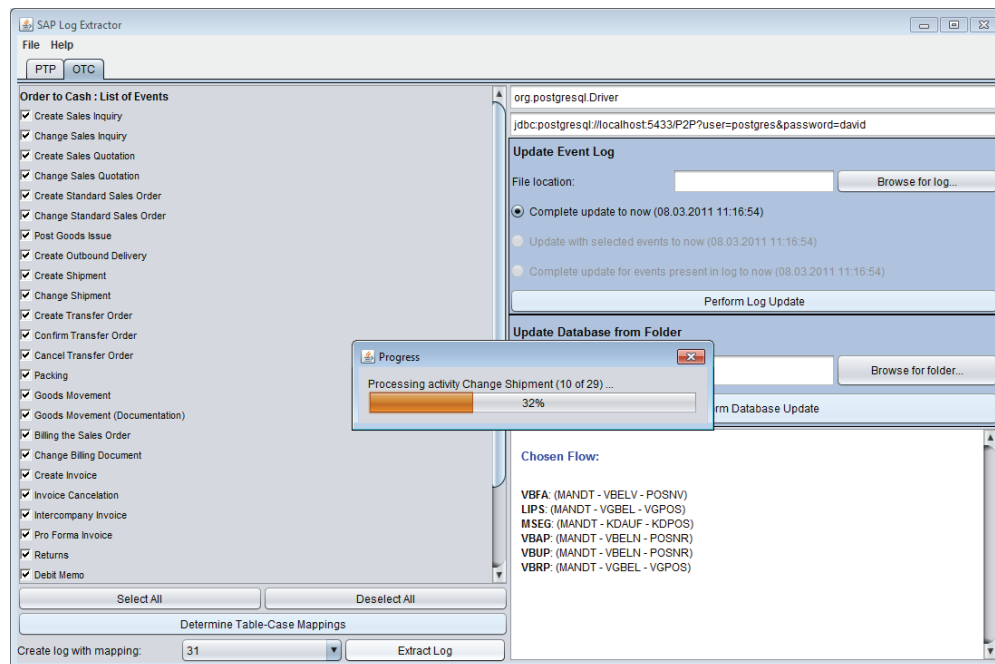


Figure 9.9: OTC Prototype Extraction

The resulting event log contains 20 different activities, containing **66,710** events spread over **14,462** cases. The timestamp of the first event is Nov 29, 1994 11:41:10 AM, while the

last event is performed during this thesis: Feb 2, 2011 1:06:33 PM. We thus have fewer events in our event log as the PTP process, Figure 9.10 gives the number of events per activity.

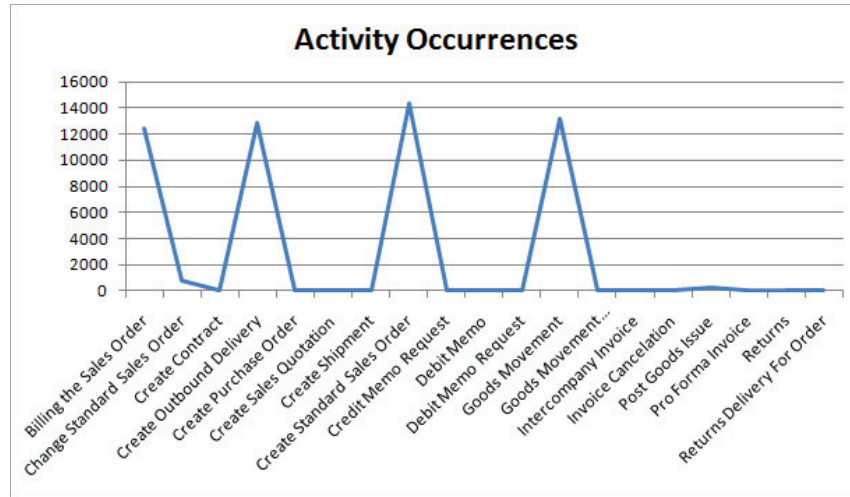


Figure 9.10: OTC Events per Activity

We can clearly see that there are four activities that have a much higher frequencies than other activities. The number of events for the activities *Billing the Sales Order*, *Create Outbound Delivery*, *Create Standard Sales Order* and *Goods Movement* stand out compared to other activity. When mining this event log and discovering the process we immediately see these four activities back in the main flow of activities (Figure 9.11). Figure 9.12 presents the model where 99% of the cases are included, this is again pretty structured. The model is created from 14318 cases (99%) and fits 14331 cases (99%) out of 14462 cases. Mining the model on 100% of the cases again results in a spaghetti-like model.

In size and understanding the sequence of activities, it is easier to set up an extraction for the Order to Cash process from SAP than the PTP process. However, the structure of the tables does not allow us much variation in retrieving a common case notion for the entire process. The reason is that there are two different ‘documents’ that play an important role in this process: the sales order document and the delivery document. Activities in this process are often related to one of these document types, creating a common link between all activities is possible, but the relations that can be extracted from SAP for example do not allow us to extract on a *sales order level*. Our conclusion in the next section generalizes this remark and discusses how to deal with this.

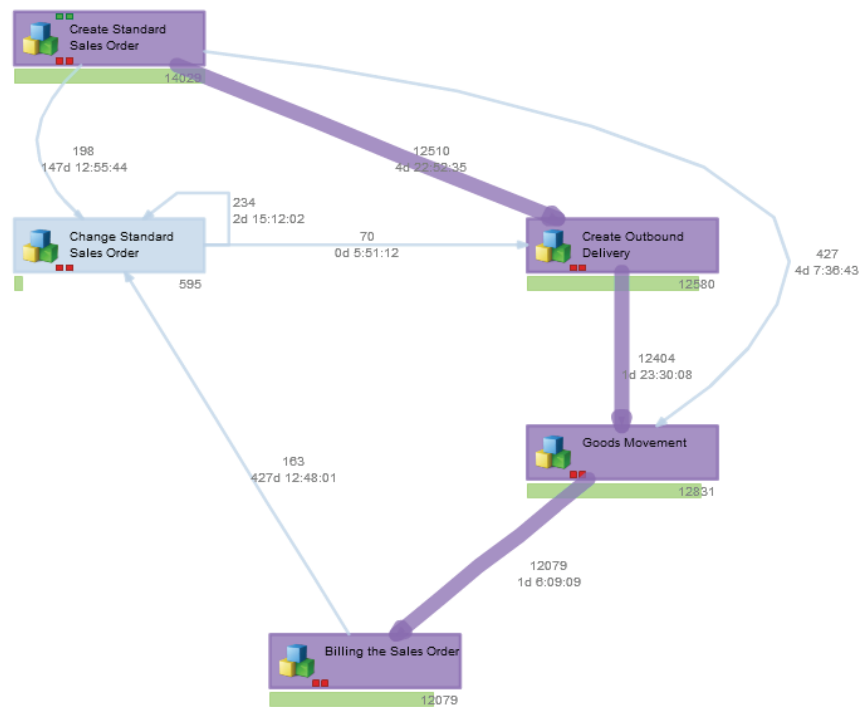


Figure 9.11: Exploring 97% of the Cases

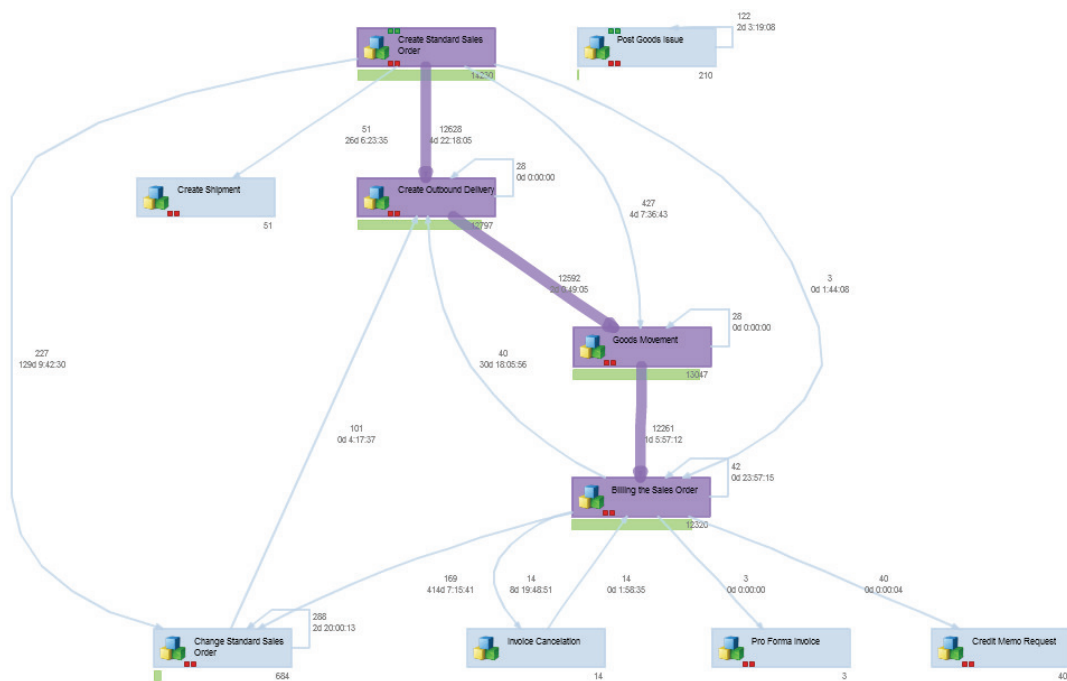


Figure 9.12: Exploring 99% of the Cases

9.3 Conclusion

In this chapter we showed the validity of our prototype by performing two case studies on processes that are implemented in our prototype: the PTP and OTC process. These are two of the most common SAP business processes. The PTP process was analyzed on three levels by using different table-case mappings and sets of activities. Furthermore we performed an incremental update of an event log for this process. The entire OTC process was analyzed once on sales order item level. For both processes we showed the characteristics of the event logs, and the models we can discover by using Reflect. As the actual mining of processes is not part of this master project, we did not analyze the processes in detail.

In general, once a process is implemented in our prototype, we have shown that it can be analyzed on different levels. The event logs we construct are influenced by the configuration of our process repository, as well as the set of activities and table-case mapping chosen through the GUI of the prototype.

The success in finding a table-case mapping for a set of activities in a business process is however dependent on the relations that exists between the involved tables. At the moment we use the relations that can be retrieved from our Repository Information System. For the OTC process we for example did not find a table-case mapping on Sales Order Document level. This could be solved by manually adding relations to our (in this case) `OTCrelations.csv` file. In general, the possibilities our approach (prototype) provides are maximized by having all possible relations between tables stored in the process repository. This same idea holds when the prototype is used on other relational databases.

Chapter 10

Conclusions

This master thesis presented the results of my master project: performing research on event log extraction from SAP ECC 6.0. The growing popularity of process mining and the fact that SAP ECC 6.0 does not provide suitable logs for process mining was the driving factor behind this research. We reflect the outcomes of this project by reconsidering the goal that was stated in the introduction: *Create a method to extract events logs from SAP ECC 6.0 and build an application prototype that supports this.*

The first contribution we made was analyzing different approaches to **extract data from SAP**. The IDoc approach appeared to be promising with respect to the updating of event logs; unfortunately it required too much customization on the target SAP system. Communication channels could be set up and configured between an extraction application and SAP, such that continuous event log extraction, and thus monitoring of processes, could be possible. However, due to the constraints this method prescribed, we chose to extract our data directly from the SAP database and store in a local database.

The method to **transform the extracted data into an event log** is another important contribution in this project. It concerns the first part of our goal and can be divided into a **preparation and extraction phase**. The preparation phase consists of selecting the activities in a business process, mapping out the detection of events in SAP and specifying the attributes to include in the event log. Its aim is to create insight in an SAP business process and where the content for the event log can be found. The extraction phase starts with selecting activities to extract, to specify the activities that should be considered within the process. This is followed by selecting the case to determine the view on the business process. If the case is known, we set up a connection with the SAP database and start constructing the event log in Futura's CSV event log format. In the construction of this method we gave a lot of practical information; i.e. where to find information necessary to perform event log extraction from SAP. Furthermore, the main steps in our event log extraction method could be applied to other ERP systems that rely on an underlying relational database as well. These represent common steps in an event log extraction procedure, the difference lays in the actual implementation of each step.

Within this procedure we proposed a method to automatically construct a case notion from a set of activities, the computation of **table-case mappings**. These table-case map-

pings enable us to tackle a common problem with data-centric ERP systems like SAP: *the determination of the case*. Having one case (where all events are instances of) unavoidably leads to some problems; the resulting issues of convergence and divergence were explained, as well as current research and opportunities to tackle these problems. Our table-case mappings are representations for cases that can be *identified by different fields in different tables*. This approach is also not limited to SAP ERP systems, but could be applied to other ERP systems that rely on an underlying relational database as well. A precondition for this is that the relations (foreign keys) between database tables are retrievable, and that subsequent activities to other objects in a process can be traced back (linked) to previous objects. In our approach we do not assume that specific SAP properties should thus hold, the approach can be *generalized* to information systems that have an underlying relational database.

The next important contribution we made concerned the **updating of events logs**. This is an entirely new extension and was shown to be feasible in SAP ECC 6.0. The approach we proposed stressed the importance of timestamps and can be executed repeatedly to perform the updating of events logs in an incremental way.

To support and validate all of the above we have *developed an application prototype*. This concerns the second part of our goal and demonstrates the applicability of our proposed solution. We can again identify a preparation and extraction phase, but have an additional update phase which can be repeatedly performed. The preparation phase ensures the creation of process repositories. These have to be created **once** for each SAP process, per type of project, and contain information necessary to perform event log extraction for that process. The extraction phase can be performed **repeatedly** once the process repositories have been set up. In the extraction phase we automated the determination of possible table-case mappings through the GUI. The user has to choose one of the proposed table-case mappings. The prototype automates the actual event log extraction as well by accessing the process repositories and communicating with the SAP database. We concluded by presenting two case studies on processes that are configured in our prototype as a proof of concept. Event logs on different levels were extracted for the Purchase to Pay and Order to Cash process.

Through the addition of the prototype we more or less have implemented an extract, load and transform approach. A method was set up to extract the data from SAP, our prototype subsequently loads this data and transforms it to an event log. Although it will remain difficult to perform process mining on data-centric ERP systems like SAP, applications can be developed that smoothen the performing of this technique. Getting acquainted with SAP, automating several important steps and the development of the table-case mapping approach are the key points of our method.

10.1 Future Work

A master project is never finished however and there is room for *improvement*. Future work might focus on the following three items:

- If emerging process mining techniques for *artifact-centric process models* become more mature, the determination of a case throughout an SAP process could be reviewed. Artifact-centric process models show good perspective on reducing issues that occur

when performing process modeling and mining for traditional data/object focused systems. However, research on this topic is still ongoing, and mining algorithms and support in process mining software still has to be created. Future research on process mining in SAP should therefore have a stronger focus on these issues, and investigate the possibility of applying an artifact-centric approach to process modeling and mining in SAP further.

- The *automated discovery of events* by checking for patterns, focussing on timestamps, in the SAP database. There are thousands of timestamps in the SAP database; an approach could be developed that does not know what activities exists in a process, but discovers, interprets and extracts occurrences of new activities. Another similar method entails the performing of an SQL trace during execution of an activity; in depth analysis of the sequence of SQL statements performed could provide knowledge in how to detect activity occurrences.
- The *incremental update* approach was proven to be valid for the processes that were implemented in the prototype. However, because this is the first attempt in updating at the event log level, this approach *could be tailored further*. Most improvements (see Section 8.5) are on an implementational level; a conceptual improvement would be to generalize this approach and remove the assumptions we had to make.

Bibliography

- [1] W.M.P. van der Aalst, A.J.M.M. Weijters, L. Maruster. *Workflow mining: Discovering Process Models from Event Logs*. IEEE Transactions on Knowledge and Data Engineering, 16(9), 1128-1142, 2004.
- [2] W.M.P. van der Aalst, R.S. Mans, N.C. Russell. *Workflow Support Using Proclets: Divide, Interact, and Conquer*. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 32(3), 16-22, 2009.
- [3] K. Bhattacharya, C. Gerede, R. Hull, R. Liu, J. Su. *Towards Formal Analysis of Artifact-Centric Business Process Models*. International Conference on Business Process Management (BPM 2007), volume 4714 of Lecture Notes in Computer Science, pages 288-304. Springer-Verlag, Berlin, 2007.
- [4] J.C.A.M. Buijs. *Mapping Data Sources to XES in a Generic Way. Master's thesis*. Eindhoven University of Technology, 2010.
- [5] T. Curran, G. Keller, A. Ladd. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Enterprise Resource Planning Series, Prentice Hall PTR, Upper Saddle River, 1997.
- [6] B.F. van Dongen, A.K. Medeiros, H.W.M. Verbeek, A.J.M.M. Weijters, W.M.P. van der Aalst. *The ProM Framework: A New Era in Process Mining Tool Support*. Applications and Theory of Petri Nets 2005. Lecture Notes in Computer Science, Volume 3536, 2005.
- [7] M. Dumas, W.M.P. van der Aalst, A.H.M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Wiley & Sons, Chichester, 2005.
- [8] D. Fahland, M. de Leoni, B.F. van Dongen, W.M.P. van der Aalst. *Behavioral Conformance of Artifact-Centric Process Models*. Eindhoven University of Technology, 2011.
- [9] Gartner. *Business Process Management Cool Vendors Report*. 2009.
- [10] M. van Giessel. *Process Mining in SAP R/3. Master's thesis*. Eindhoven University of Technology, 2004.
- [11] C.W. Günther. *XES: Extensible Event Stream Standard Definition*. Fluxicon Process Laboratories, November, 2009.
- [12] IDS Scheer. *ARIS Platform - System White Paper*. June, 2008.

- [13] J.E. Ingvaldsen, J.A. Gulla. *Preprocessing Support for Large Scale Process Mining of SAP Transactions*. Norwegian University of Science and Technology, 2008.
- [14] R.J.J. Kerstjens. *Process Analysis in ARIS PPM, BusinessObjects and the ProM Framework*. Master's thesis. Eindhoven University of Technology, 2006.
- [15] E. Lute. *Over Business Intelligence: Data is zilver, informatie is goud*. TIEM, 2010.
- [16] A.K. Medeiros, A.J.M.M. Weijters, W.M.P. van der Aalst. *Genetic Process Mining: An Experimental Evaluation*. Data Mining and Knowledge Discovery, v.14 n.2, April, 2007.
- [17] J. Mendling, H.W.M. Verbeek, B.F. van Dongen, W.M.P. van der Aalst, G. Neumann. *Detection and prediction of errors in EPCs of the SAP reference model*. Data & Knowledge Engineering, v.64 n.1, p.312-329, January, 2008.
- [18] SAP AG. *SAP Solution Manager: A Platform for Reducing Risk and Total Cost of Ownership*. 2004
- [19] SAP AG, Global Communications. *Annual Report 2009*. 2010
- [20] I.E.A. Segers. *Deloitte Enterprise Risk Services, Investigating the application of process mining for auditing purposes*. Master's thesis. Eindhoven University of Technology, 2007.
- [21] A. Silberschatz, H.F. Korth, S. Sudarshan. *Database System Concepts*. 4th Edition. McGraw-Hill Book Company, 2001.
- [22] W. Sun, T. Li, W. Peng and T. Sun. *Incremental Workflow Mining with Option Patterns*. International Conference on Systems, Man, and Cybernetics (SMC 2006).
- [23] H.W.M. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, W.M.P. van der Aalst. *ProM 6: The Process Mining Toolkit*. BPM 2010 Demo, September, 2010.

Appendix A

Glossary

ABAP	Advanced Business Application Programming, a programming language developed by SAP to write applications for the SAP ERP program.
Activity	An action or task that can be executed in a process.
ALE	Abbreviation for Application Link Enabling, a mechanism to exchange business data between SAP applications. ALE provides a program distribution model and technology which enables to interconnect programs across various platforms and systems.
Case	An object that passes through a process. Examples are persons, purchase orders, complaints etc.
Case Identifier	A unique identifier that identifies a specific case.
Configuration	Configuration of SAP to enable the execution of certain business processes. It is the process of tailoring SAP software by selecting specific functionality from a list of those supported by the software, very much like setting defaults. Each SAP instance can be distinctively configured to match the needs and desires of the customer (with limits).
CSV	The Comma-Separated Values file format is a file format used to store tabular data in plain textual form that can be read in a text editor. Lines in the CSV file represent rows of a table, and commas in a line separate what are fields in the table's row.
Customization	Making changes to SAP's underlying ABAP source code in order to fulfill industry-specific demands that cannot be covered by SAP's basic functionality.
EDI	Abbreviation for Electronic Data Interchange.
Event	An occurrence of an activity.
GUI	Graphical User Interface.
IDoc	Intermediate document, the container for application data in the SAP ALE system.
Process Instance	An instance of a 'case' in a process.

SAP JCo	SAP Java Connector is a middleware component that enables the development of SAP-compatible components and applications in Java. It supports communication with the SAP Server in both directions: inbound calls (Java calls ABAP) and outbound calls (ABAP calls Java).
Referential Integrity	Referential integrity is a database concept that ensures that relationships between tables remain consistent. When satisfied, it requires every value of one attribute (column) of a relation (table) to exist as a value of another attribute in a different (or the same) relation (table).
RFC	Abbreviation for Remote Function Call, the standard SAP interface for communication between SAP client and server over TCP/IP or CPI-C connections.
Table-Case Mapping	A mapping of tables to a couple of fields that together identify a case.
XES	An open standard for storing and managing event log data, see http://code.deckfour.org/xes/ .

Appendix B

Downloading Data from SAP

Caution must be taken when specifying the download format and file type in order to retain specific data formats. If a table is downloaded in Spreadsheet format as an MS Excel file, MS Excel puts all data in a *general* format. Although this is correct for most data, it for example gives problems for fields that contain keys that are composed of multiple values or that contain large numbers. An example of a composed key is the field *TABKEY* in table CDPOS. Putting this into a general format removes leading zeros from the key, messes up the structure of the key and prevents us from retrieving specific parts of the key. the *TABKEY* presented below is an example of this.

$$TABKEY(090001000099200010) = \underbrace{090}_{MANDT} \underbrace{0010000992}_{BANFN} \underbrace{00010}_{BNFPO}$$

MS Excel would round this number to 900010000992**000000**. This way we can not retrieve the BNFPO number (line item number) of an order or requisition. When fields like *TABKEY* are present, the best option is to download the table from SAP in Spreadsheet format as a CSV file. This gives unformatted data and if the data needs to be displayed in MS Excel, use the data import on this CSV file and specify that all columns should be treated as *Text*.