Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Architecture of Information Systems Research Group

Mapping Data Sources to XES in a Generic Way

Master Thesis

ing. J.C.A.M. Buijs

Supervisors:
prof.dr.ir. W.M.P. van der Aalst
dr.ir. H.M.W. Verbeek
dr. G.H.L. Fletcher

Final version

Eindhoven, March 2010

Abstract

Information systems are taking a prominent place in today's business process execution. Since most systems are complex, enterprise-wide systems, very few users, if any, have a clear and complete view of the overall process. In the area of process mining several techniques have been developed to reverse engineer information about a process from a recording of its execution. To apply process mining analysis on process-aware information systems, an event log is required. An event log contains information about cases and the events that are executed on them.

Although many systems produce event logs, most systems use their own event log format. Furthermore, the information contained in these event logs is not always suitable for process mining. However, since much data is stored in the data storage of the information system, it is often possible to reconstruct an event log that can be used for process mining. Extracting this information from the business data is a time consuming task and requires domain knowledge. The domain knowledge required to define the conversion is most likely held by people from business, e.g. business analysts, since they know or investigate the business processes and their integration with technology. In most cases business analysts have no or limited programming knowledge. Currently there is no tool available that supports the extraction of an event log from a data source that doesn't require programming.

This thesis discusses important aspects to consider when defining a conversion to an event log. The decisions made in the conversion definition influence the process mining results to a large extend. Defining a correct conversion for the specific process mining project at hand is therefore crucial for the success of the project. A framework to store aspects of such a conversion is also developed in this thesis. In this framework the extraction of traces and events as well as their attributes can be defined. An application prototype, called 'XES Mapper' or 'XESMa', that uses this conversion framework is build.

The XES Mapper application guides the definition of a conversion. The conversion can be defined without the need to program. The application can also execute the conversion on the data source, producing an event log in the MXML or XES event log format. This enables a business analyst to define and execute the conversion on their own. The application has been tested with two case studies. This has shown that many different data source structures can be accessed and converted.

Keywords: data conversion, database, event log, process mining, process-aware information system

Preface

This master thesis is the result of my graduation project which completes my Business Information Systems study at Eindhoven University of Technology. The project was performed internally at the Architecture of Information Systems group of the Mathematics and Computer Science department of Eindhoven University of Technology. The project also concludes an education career of many years. Especially the last two years were particularly interesting and motivating. And although the future is unknown, the fundament is very good.

First of all I would like to thank Wil van der Aalst for providing this opportunity and for his guidance during my master's project. I would also like to thank Eric Verbeek for his continuous help and optimism. Furthermore my thanks go out to George Fletcher for providing valuable feedback.

I would also like to thank Christian Günther for sharing his expertise with me. And without Melike Bozkaya I wouldn't be doing this master project. Thanks also go out to Joost Gabriels from LaQuSo for helping me with the SAP case study and for the many nice train commutes. I would also like to thank all the colleagues at the department and especially Maja for the many nice lunches, meetings and conversations.

And of course, my thanks also go out to my mother, father and two brothers. Without them I would not have come this far. The same can be said for my girlfriend Debbie who has supported and motivated me all the way. Special thanks go out to Thijs for being my college-buddy and for making me exceed my expectations.

Joos Buijs
March 2010

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

This master thesis is the result of the graduation project for the Business Information Systems master at Eindhoven University of Technology (TU/e). The project is carried out within the Architecture of Information Systems (AIS) group of the Mathematics and Computer Science department of TU/e. The AIS research group investigates process-aware information systems and the business processes they support. The group is world leading in the field of process mining which allows for the analysis of the process execution based on event logs generated by the information system.

Event logs are not always available but event related information is often stored in the database of the information system. This thesis investigates the problem of converting data sources to an event log format. Special attention is paid on using this event log in process mining. This resulted in a conversion definition framework which allows a business analyst to specify the extraction. A solution has been realized as a Java application prototype where one can define and execute the conversion on a data source. We have also shown the viability of our approach with two case studies.

This chapter introduces the before mentioned problem in more detail. In Section 1.1 the context of this master thesis is explained. Section 1.2 discusses the problem description which is followed by the research objective in Section 1.3. Then in Section 1.4 the scope of the research is defined. To conclude this chapter the research method and thesis outline is provided in Section 1.5.

1.1 Thesis Context

Information systems are taking a prominent place in today's business process execution. Hospitals implement patient tracking systems and document patient, decease and treatment information. Municipalities record information about civilians and provide access to it for other institutions. One of the best known examples of information systems widely used in industry are the so-called Enterprise Resource Planning (ERP) systems such as SAP and Oracle, etc. These systems manage nearly anything that happens within a company, be it finance, human resources, customer relationship management or supply chain management. In order to support the business process these systems are becoming more and more process-aware. They delegate tasks to groups of employees and keep track of everything that happens within the system.

Since most systems are complex, enterprise-wide systems, very few users, if any, have a clear and complete view of the overall process. And even if the business process is very well documented, the system and its users might deviate from it, be it unknowingly, by accident or on purpose. This might be caused by system configuration errors due to the size and complexity of most systems. Fraud is another problem which can result in great financial costs [20, 23]. Companies are also required nowadays to audit their IT systems regularly for evaluating the implementation of security protocols, IT governance and accounting compliance regulations. Furthermore, if a company wants to improve or redesign its business process, a good starting point would be to look at the execution

of the current process.

In the area of process mining several techniques have been developed to reverse engineer information about a process from a recording of its execution. This recording, known as an event log, lists all events that are executed on a certain case together with information such as when it was executed and who executed it. Process mining techniques exist to create a process model based purely on the information contained in such an event log. Other process mining analysis techniques include process performance visualization, process conformance checking, social network analysis, business rule detection, rule verification and case prediction based on historical data.

To provide a platform for the implementation and execution of all these algorithms the ProM framework has been developed at the Architecture of Information Systems research group at Eindhoven University of Technology. This framework currently includes over 280 plug-ins that each contribute to the analysis spectrum of process mining. Process mining and the ProM framework have proven its viability and applicability in the last years [3, 13, 21, 22, 23].

In general, process mining can only be applied if a clear event log is available to start the analysis from. As mentioned before, this event log should contain information about cases and the events that are executed on them. The ProM framework supports the MXML event log format. Starting from version 6 (to be released) the new XES event log format is also supported. Both these formats store information of cases with their events. Although many systems produce event logs, most systems use their own event log format. Furthermore, the information contained in these event logs is not always suitable for process mining. SAP for instance does not produce event logs suitable for process mining [13, 19]. Even though the information is not directly available in the event log, in most cases it is present in the data storage of the information system. The problem is that the required information is hidden within the data in a structure that is focussed on correct and efficient data storage for the systems' purpose.

1.2 Problem Description

As discussed in the previous section, event log data is not always present in a format that can be used for process mining. Since much data is stored in the data storage of the information system, it is often possible to reconstruct an event log that can be used for process mining. The data necessary to reconstruct an event log is hidden between all the other business data. Think for instance of the activity 'create order' in an ERP system where the date and time of creation of the order and the user who created it might be stored with the order data itself but not in the event log.

Extracting this information from the business data is a time consuming task and requires domain knowledge. To complicate things even further, the conversion to the target event log format is not always clear. Decisions on how and what to extract from the data source to the event log can influence the process mining results drastically during the analysis phase. Most projects focus on a certain part of the process so extra attention should be paid to the events occurring in that particular part of the process. This also requires knowledge of the business process and what events to search for in the data model. All this makes it very complex and time consuming to prepare an event log to be used for process mining.

The domain knowledge required to define the conversion is most likely held by people from business, e.g. business analysts, since they know or investigate the business processes and their integration with technology. Although business analysts often operate in the field between business and IT, most business analysts don't have much programming experience and in general don't know how to construct an event log from data dispersed in the data storage of the system.

Converting data to a fixed event log format is not a new problem within the area of process mining. For this purpose the ProM Import Framework was created. This framework can convert data to the MXML event log format, as used by ProM. Since the ProM Import Framework makes use of plug-ins written in Java to convert data from a certain data source to the MXML format, it requires the process miner, or anyone else, to program a Java plug-in to perform the conversion. Although there is a collection of plug-ins for various systems and data structures, chances are

that a new plug-in needs to be written. The main problem with this approach is that it requires the domain expert to create a piece of Java code to perform the conversion. Furthermore, in a plug-in the conversion logic itself is hidden within the code. This makes it rather unclear to see how the actual conversion is specified. Also, the plug-in does not provide much guidance, beyond the framework structure, to guide in the definition of the conversion from the data source format to the event log format.

The problem tackled in this thesis can be summarized as follows in the problem statement of this project:

> ***Problem Statement**: There is no tool that guides a business analyst towards converting data from a data source to an event log format suitable for process mining without the need to program.*

1.3 Project Goal

The previous section made clear that currently there is no tool that supports a business analyst in converting data from a data source to an event log format without the need to program. Briefly, the goal of this project therefore is to create such an application. This section elaborates on the project goal and defines the project goal in one sentence.

The application should be able to use data sources in a relational database format. A relational database consists of tables which consist of rows and columns of information. Since most information systems use relational databases for their internal data storage this choice enables source data from many systems to be used as input. By supporting this input format no manual intermediary conversions or extractions need to take place and the application will be able to read directly from (a copy of) the original data source as used by the information system. Furthermore, there are many libraries available to connect to relational data sources of various brands and types and therefore nearly any relational data source can be used as input for the application. Note that there are libraries available that enable comma-separated-value files and XML files to be accessed as if they were relational databases. *Therefore, by choosing relational database systems as our data source input format we enable access to a wide variety of information system data storages.*

A logical choice for an event log format to support would be the MXML format discussed in Sections 1.2 and 2.2.1 which is well known by process miners. However, a new format is currently under development by the IEEE Task Force Process Mining[1] to serve as a standard in logging event data. This new format XES is developed by the Architecture of Information Systems research group at the Eindhoven University of Technology in cooperation with Fluxicon[2]. This new event log format solves some issues encountered while using MXML for recording real life process executions and allows for more flexibility. Furthermore, it is still possible to generate event logs in the MXML format under certain conditions. More about the event log format XES can be found in Section 2.2.2.

There are some aspects to focus on during development of the application. The first aspect to focus on is to create an application that will be able to convert many data structures encountered within relational databases. Since some data models defined within relational databases can be quite complex the application should be able to handle these complexities. At the same time the application should enable users to define a conversion while limiting the amount and degree of programming required. To further aid the user in defining and checking the conversion a visualization of the conversion definition from the data source to the event log format should be made. This will also aid in explaining the conversion to other stake holders for verification. Together these three aspects will be the focus of our application development.

The project goal as described above can be summarized as follows:

> ***Goal of the project**: Create an application prototype that helps a business analyst*

[1] See `http://www.win.tue.nl/ieeetfpm`
[2] See `http://www.fluxicon.com`

to define and execute a conversion from a data source in table format to the event log format XES requiring as little programming as possible.

1.4 Research Scope

The scope of the application is to enable the definition and execution of a conversion from any data source to an event log. As discussed in the previous section the application will be able to use tables from a relational databases as input. Data is in many cases stored in relational databases or could be converted into such a structure. The output format will be an event log in the XES format. This is summarized and visualized in Figure 1.1.

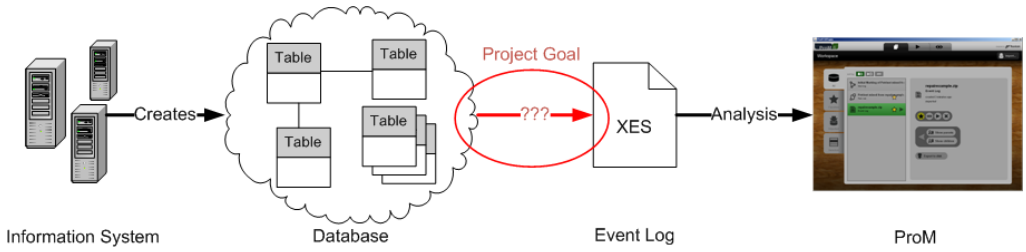


Figure 1.1: Project scope

The research in this project will briefly discuss the information systems generating the data. Implications of some of the conversion decisions made by the user on the process mining analysis results will also be discussed. The project will not focus on data analysis, data checks or other functionality beyond converting data to the event log format following the mapping provided by the user. ProM is the application designed for these kind of tasks.

1.5 Research Method and Outline

To satisfy the project's goal and solve the problem statement, the following steps are taken:

Perform a literature study on relevant concepts
The first step is to investigate existing literature on topics such as the information systems and their data structures that are used as data source. Also event log formats, process mining and existing tools are investigated. The findings of this literature study are presented in Chapter 2.

Conduct a study on data conversion requirements
Chapter 3 discusses the decisions that should be made during the conversion specification and the impact of these decisions. Topics discussed are the selection of the case to follow and which events and attributes to include. The problem that there might not always be a direct and single relation between traces and events is also discussed.

Specify a general conversion framework to define a conversion
The next step is to define a framework for defining the conversion which is presented in Chapter 4. This chapter discusses the information required for a conversion and how this is implemented in the conversion framework. The way of visualizing the conversion specification is also discussed.

Implement the conversion framework into an application to define and execute it
Chapter 5 explains how the conversion framework from Chapter 4 is implemented in an application. The technical design decisions and internal design are also discussed here.

Test the application and conversion framework on case data
The validity of the approach is demonstrated by performing several case studies which are discussed in Chapter 6. This chapter shows all the steps that need to be taken to extract the required information from the data source to create an event log suitable for process mining.

Conclude
This thesis is concluded by Chapter 7 in which the entire approach is evaluated and is argued whether or not the project achieved its goal. This chapter also includes a discussion of future work aiming at solving existing problems or improving the results of the project.

Chapter 2

Preliminaries

This chapter introduces preliminary concepts used throughout this thesis. Section 2.1 starts with a brief discussion on information systems and the way they store process related data. This includes an example of how events of a procurement process are stored in an SAP system in Section 2.1.1. Section 2.2 introduces the concept of event logs in general and looks at two specific formats. In Section 2.3 process mining and the process mining framework ProM are introduced which analyze process information recorded in event logs. This chapter is concluded with Section 2.4 in which applications that could also be used to solve the problem stated in the previous chapter are briefly discussed.

2.1 Process-Aware Information Systems

Information systems are more and more supporting the execution of (business) processes. In [11, p. 7] a Process-Aware Information System (PAIS) is defined as follows:

> *A Process-Aware Information System is a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models.*

Process-aware information systems are mainly used to support the execution of business processes. Within a PAIS the process definition itself is not embedded deeply within the code of the application. The process is defined as a model that can be viewed and changed. This allows for the modification of the process definition without the need to modify the code of the application. Changes in the business process might be triggered by process changes in the environment of the company or by internal process improvement drivers. Furthermore, PAISs allow for automatic enactment of the processes supported by an organization. Information can be automatically routed to the appropriate human actors or applications.

Process supporting systems might be generic as is the case for workflow management systems (WfMS) [2]. These systems don't incorporate information about the structure and processes of any particular organization. Instead these systems allow for the definition of processes, applications, organizational entities and so on. The WfMS then 'orchestrates' the process by triggering workflows and enabling one or several tasks. These tasks are then routed to people or applications who/which complete them. As tasks are completed the WfMS continues according to the process specification by dispatching subsequent tasks until the process is completed for that case.

Other systems implement process-awareness by supporting it from within their specific application. Examples are collaboration tools, project management tools and case handling systems. Another commonly used type of PAIS are Enterprise Resource Planning (ERP) systems which support all the business processes of a company such as supply chain management, CRM and human resources. For instance SAP, Microsoft Dynamics and Agresso Business World [26] have

process models or reference models to guide the process. However, these process models are not always enforced to control the process and allow the user some form of freedom [11, p. 236].

Nearly all systems record event related data in one way or another. Be it in special purpose event logs (see Section 2.2) or 'hidden' between the data in their data storage. Section 2.1.1 describes the way SAP records data related to events occurring in the procurement process as an example.

2.1.1 Event Data in SAP Systems

This section demonstrates how event related information can be extracted from the data storage of SAP. SAP is one of the most used Enterprise Resource Planning (ERP) systems. As shown in [19, 13] SAP does not produce event logs that are suitable for process analysis via process mining (see Section 2.3). However, it is possible to reconstruct an event log suitable for process mining by extracting the correct information from the database used by SAP to support the process.

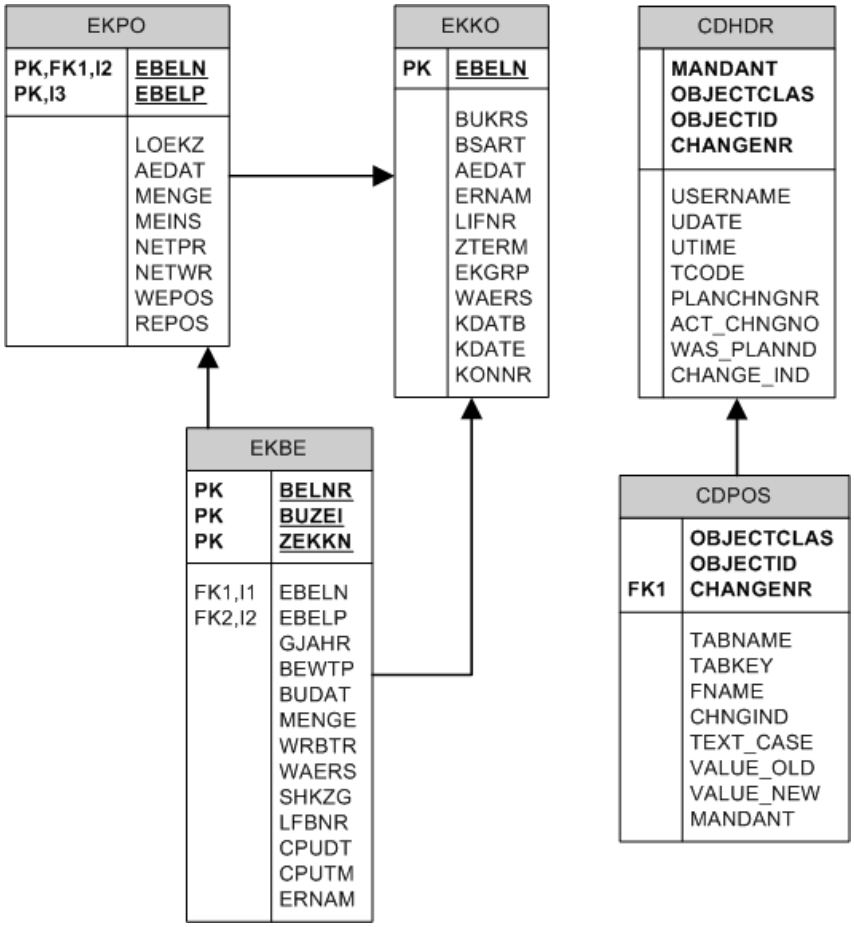


Figure 2.1: Procurement part of an SAP data model

Figure 2.1 shows a selection of tables and their fields from the data storage of a SAP system. The tables and fields are related to the procurement process. Please note that the relationships between the tables were added manually. The SAP database does not contain any relationships as these only exist in the application layer. The value 'PK' before a field indicates that these fields together form the primary key of that table. They uniquely identify each record. The fields in bold are mandatory fields for that table. Whenever a field is preceded by a 'FK' indicator it means that this field refers to (one of) the primary keys of another table.

The table *EKPO* stores order line information. Each order line is related to an order as stored in table *EKKO*. Each order has a unique number stored in the *EBELN* field of the *EKKO* table. Each order line is identifiable by a combination of the order identifier (*EBELN*) and a separate order line identifier stored in the *EBELP* field of the *EKPO* table. The *EBELN* field of the *EKPO* table refers to the order in the *EKKO* table it belongs to. Other attributes included in the *EKPO* table for order lines are the unit (in the field *MEINS*) and the value (*NETWR*) of the order line.

The *EKBE* table stores the history of purchase documents containing information related to the handling of goods and invoices related to the orders.

The data stored in the SAP system allows us to extract information about the event of creating a new purchase order. This includes the timestamp of creation and the user who created it. The date of creation of a new purchase order is stored in the field *AEDAT* of the *EKKO* table. The user who created it is stored in the *ERNAM* field of the *EKKO* table. Additional information could be retrieved from the data storage but the event name, date and time of execution and the user who performed the action suffices for now.

In a similar way a change of one of the order lines after creation can be detected by looking at the changes on objects. Changes are recorded in the tables *CDHDR* and *CDPOS*. The *CDHDR* table contains the change header with general information of a change. The *CDPOS* table records the detailed changes made on objects. Each change header can contain multiple detailed changes. For each detailed change the table and field that is changed is stored. Since order lines are stored in the *EKPO* table we search for changes on this table. We select all the change records where the *TABNAME* field of the *CDPOS* table refers to the *EKPO* table. To connect this change to an order line, we use part of the key stored in the *TABKEY* field. This field stores a concatenation of certain fields, depending on the table. In this case a part of the *TABKEY* is a combination of the *EBELN* and *EBELP* fields of the *EKPO* table. The date of the change is stored in the *CDHDR* table in the field *UDATE* and the time in the *UTIME* field. The detailed change record in the *CDPOS* table can be connected to the change header via the *CHANGENR* fields of both the *CDPOS* and the *CDHDR* tables. In the change header (*CDHDR*) table the user who performed the change is stored in the field *USERNAME*. As attribute of the change event the value before the change can be added by extracting it from the *VALUE_OLD* field. Similarly the new value can be recorded by using the *VALUE_NEW* field. Additional attributes can be added if desired.

Events such as the receipt of the goods and payment of the invoice can be extracted in a similar way. This can be done by using the purchasing history as stored in table *EKBE*.

This section shows that we can reconstruct events occurring for a certain case by using data from the central data storage of the system. However, one needs to know where the information is stored and how to extract it. Chapter 3 discusses what information needs to be present in the data storage and how to extract it to construct an event log.

2.2 Event Logs

In general an event log records the events that occur in a certain process for a certain case. Many of the information systems as discussed in the previous section log events in an event log. Figure 2.2 visualizes the general structure of event logs and their relation to the definition of a process with activities. The *process definition* specifies which *activities* should be executed and in which structure. When a new case is started a new instance of the process is generated which is called a *process instance*. This process instance keeps track of the current status of the case in the process. This process instance might leave a *trace* of *events* that are executed for that case in the event log. Each event is an instance of a certain activity as defined within the process definition. Furthermore, events are ordered to indicate in which sequence activities have occurred. In most cases this order is defined by the date and time or *timestamp* attribute of the event. Sometimes the start and stop information is recorded of a single activity. This is recorded in the *event type* attribute of the event. 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 related to the event for example the data attributes added or changed

in the activity.

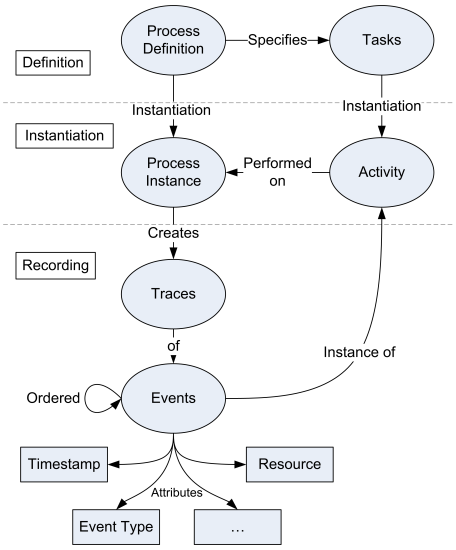


Figure 2.2: General event log structure.

Unfortunately, not every information system logs events in the described way. Information about the relation between events and activities or even between traces and process instances is often not recorded. The main reason for this is that the event log is not seen as a recording of the execution of a process by system designers. In some systems the event log is used for debugging errors encountered in the system. In other systems events need to be derived from data scattered over various tables as shown in the SAP example in Section 2.1.1.

However, even if the event log (partly) follows the described structure the format of these event logs still differs between systems. Before process analysis techniques such as process mining (see Section 2.3) can be applied on these event logs they should be converted to a standardized format. For this purpose the Architecture of Information Systems research group at Eindhoven University of Technology specified the MXML event log format [9] which is discussed in Section 2.2.1.

The MXML format has proven its use as a standardized way to store event logs for use in process mining. However, almost no information system in practice record their event logs directly in this format. During the use of the MXML standard several problems with its way of storing event related data have been discovered. One of the main problems is the semantics of additional attributes stored in the event log. These are all treated as string values with a key and have no generally understood meaning. Another problem is the naming given to different concepts. This is caused by the assumption of MXML that strictly structured processes would be recorded in this format [14, p. 49].

To solve the problems encountered with MXML and to create a standard that could also be used to store event logs from many different information systems directly a new event log format is under development by the IEEE Task Force Process Mining[1]. This new event log format is named XES and stands for eXtensible Event Stream. The format is discussed in detail in Section 2.2.2. The solution developed in this project supports this format directly but is also able to create an MXML event log.

2.2.1 The MXML Event Log Format

The MXML event log format was created in 2005 by Van Dongen and Van der Aalst and is presented in [9]. The goal of the format is to standardize the way of storing event log information. The main purpose of the format is to allow the application of analysis techniques on event logs

[1] See `http://www.win.tue.nl/ieeetfpm`

```
<ProcessInstance id="Order 1" description="instance with Order 1">
  <Data>
    <Attribute name="totalValue">2142.38</Attribute>
  </Data>
  <AuditTrailEntry>
    <WorkflowModelElement>Create</WorkflowModelElement>
    <EventType>complete</EventType>
    <originator>Wil</originator>
    <timestamp>2009-01-03T15:30:00.000+01:00</timestamp>
    <Data>
      <Attribute name="currentValue">2142.38</Attribute>
      <Attribute name="requestedBy">Eric</Attribute>
      <Attribute name="supplier">Fluxi Inc.</Attribute>
      <Attribute name="expectedDelivery">2009-01-12T12:00:00.000+01:00</Attribute>
    </Data>
  </AuditTrailEntry>
</ProcessInstance>
```

Listing 2.1: Part of an MXML XML file with one trace and one event

such as process mining. By defining a standard the analysis algorithms can be based on a common input format.

Listing 2.1 shows part of an MXML event log without the header. The selection shows the definition of one process instance and one recorded event. The process instance represents one case that runs in the system and in this example it represents an order. Events are recorded in so called *AuditTrailEntry* items and in this example the entry represents the creation of a new order. Each process instance has a uniquely identifiable string which in this example is 'Order 1'. The description of a process instance can be used to further describe the process instance briefly. Process instances can contain data values which can be used during analysis to detect relationships between data values and process characteristics such as route choice or activity duration. In this example the total order value is recorded in an data attribute with the name 'totalValue'.

The event recording consists of several special attributes as well as custom data attributes. The *WorkflowModelElement* attribute stores the name of the activity that triggered the event. In the example it is the activity 'Create' that was performed. Since an event is an atomic recording of something that happens, it does not have a duration. Since activities do have a duration, event types need to be distinguished to record the different states an activity can be in. Most used types are the *start* and *complete* types which indicate when work on an activity started and when it was finished respectively. Other examples are the *suspend* and *resume* types which indicate that an activity was momentarily paused and later on continued with again. The event in the example indicates the completion of the activity 'Create'. To record who performed a certain activity the user name or identifier can be stored in the (optional) *Originator* attribute. In this case 'Wil' performed the action. Please note that besides a human user of the system this can also refer to the system itself or another, external, system that performed the activity. Another optional attribute is the *Timestamp* which stores the date and time of the event execution. In the example the event was executed on January 3, 2009 at 15:30 hours GMT+1. Event recordings can also contain additional data attributes in the same way as process instances. The example contains 'currentValue', 'requestedBy', 'supplier' and 'expectedDelivery' data attributes. The exact meaning of these attributes and their values is specified nowhere but can be deduced from the attribute keys in this case.

The header of an MXML log file can contain information on the data source and additional general data attributes. Each MXML event log can also contain recordings of multiple processes. Each of these processes consists of process instances and audit trail entries as shown in the example. A complete definition of the MXML meta model can be found in Appendix B and in [9].

```
<trace>
  <string key="concept:name" value="Order 1"/>
  <float key="order:totalValue" value="2142.38"/>
  <event>
    <string key="concept:name" value="Create"/>
    <string key="lifecycle:transition" value="Complete"/>
    <string key="org:resource" value="Wil"/>
    <date key="time:timestamp" value="2009-01-03T15:30:00.000+01:00"/>
    <float key="order:currentValue" value="2142.38"/>
    <string key="details" value="Order creation details">
      <string key="requestedBy" value="Eric"/>
      <string key="supplier" value="Fluxi Inc."/>
      <date key="expectedDelivery" value="2009-01-12T12:00:00.000+01:00"/>
    </string>
  </event>
</trace>
```

Listing 2.2: Part of an XES XML file with one trace and one event

2.2.2 The XES Event Log Format

This section discusses the XES format in depth since no official specification has been published up until now and this thesis heavily relies on the structure of this format. Please note that this thesis is based on XES definition version 1.0, revision 3, last updated on November 28, 2009. Minor changes might be made before the final release and publication of the format. More extensive and up to date information regarding this format can be found at `http://code.deckfour.org/xes`.

In order to explain the structure of an XES event log, the example from Section 2.2.1 in the MXML event log format is rewritten to fit into the XES format and is shown in Listing 2.2. Each *trace* element describes the events occurring for one specific instance, or case, of the logged process. In this example our case is an order. Every trace contains an arbitrary number of *event* objects. Our example contains only one event. Events represent atomic states of an activity that have been observed during the execution of a process. As such, an event has no duration, just like in the MXML format.

The *trace* and *event* objects do not contain any information and only define the structure of the document. Therefore each of the three elements can have one or more *attributes* which store the information of their parent. All attributes have a string based key (without whitespace characters) and a value. Attributes can be of one of the types string, date, integer, float and boolean. Attributes themselves can also have attributes defining more specific properties.

In our example in Listing 2.2 several attributes are defined. The *trace* object has two attributes 'concept:name' and 'order:totalValue'. To understand what they mean we must refer to the extension that defined them, if any. The attribute 'concept:name' is defined by the *concept* extension which is one of the standard extensions provided with the XES format. The relation to this extension is defined by the prefix 'concept:' in the attribute name. Each extension used in the event log has his own attribute prefix. The concept extension specifies that *log*, *trace* and *event* objects can have an attribute 'name' which stores the generally understood name of the element. The other trace attribute, 'order:totalValue' is defined by the order extension. This is not one of the standard extensions but extensions can be defined by anybody. In this case the order extension specifies a 'totalValue' attribute for the trace recording the total value of the order. As mentioned before, the concept extension also defines a 'name' attribute for events which refers to the name of the executed activity. In our example this is the 'create' activity. The next event attribute is the 'transition' attribute defined by the *lifecycle* extension. This attribute indicates the lifecycle transition, referred to as 'event type' in MXML, which in many cases is either 'start' or 'complete'. The *organizational* extension allows for recording the resource that executed the activity together with its role and group. In our example we only use the resource attribute to record that 'Wil' created the order. The *time* extension specifies that the 'timestamp' attribute for events record the execution date and time. In the example the event was executed on January 3, 2009 at 15:30

hours GMT+1. The *order* extension mentioned earlier also defines a 'currentValue' attribute for events which records the current value of the order. Not all attributes need to be defined by an extension. Note that the 'detail' attribute in the example does not refer to an extension. This also demonstrates the fact that also attributes can have attributes. In this example we use the 'details' attribute to group other attributes together. These attributes are 'requestedBy', 'supplier' and 'expectedDelivery'.

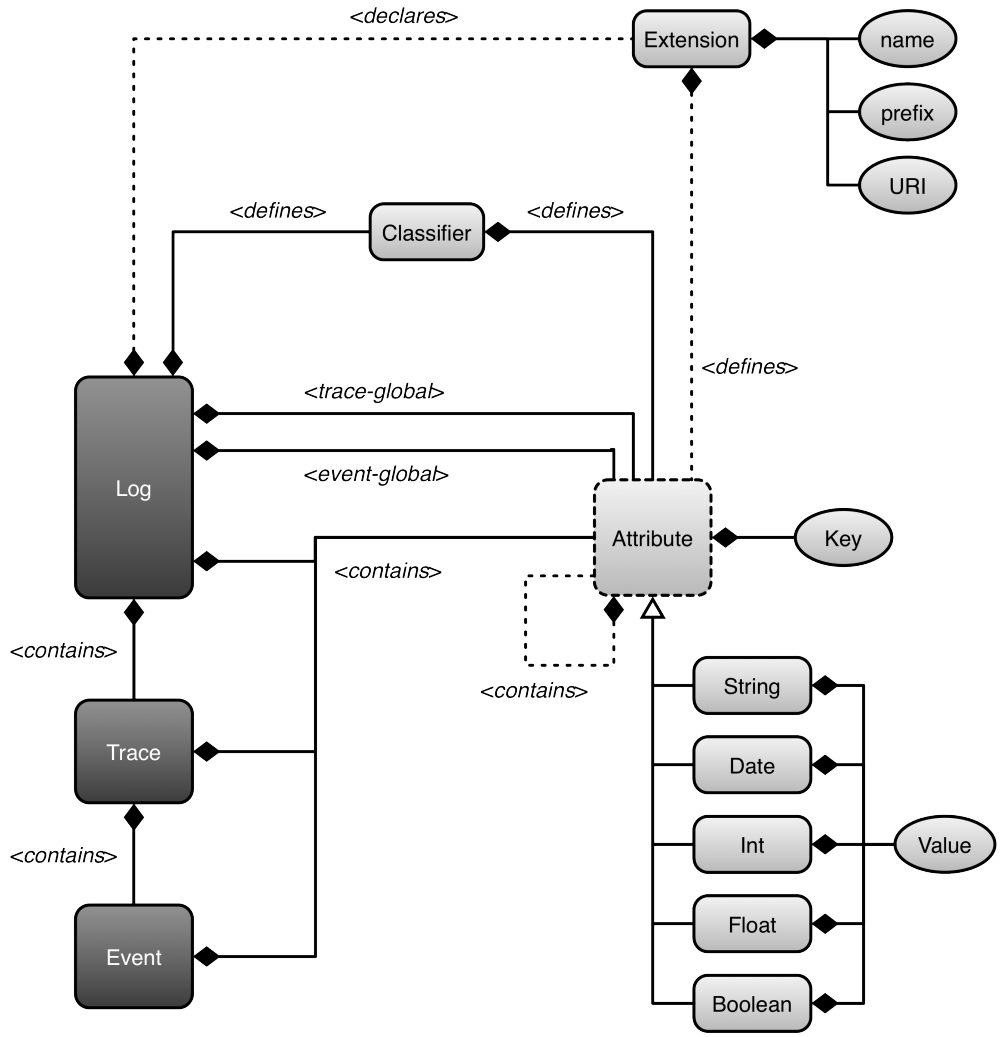


Figure 2.3: XES meta model

Figure 2.3 shows the XES meta model and is taken from [15]. The meta model shows additional properties of the XES format not discussed in the previous example. One of these properties is that the *log* object also holds a list of *global attributes* for traces and events. Including an extension in the log file does not force the use of all or any of the attributes defined. Global attributes specify what attributes are understood to be available and properly defined for each trace or event in the log. Since certain analysis methods require particular information to be available in the log, the global attributes provides an easy way to check which attributes are present.

Furthermore, *event classifiers* can be specified in the log object which assign an identity to each event. This makes events comparable to other events via their assigned identity. Classifiers are defined via a set of attributes, from which the class identity of an event is derived. Examples of classifiers are a combination of the event name and lifecycle transition as used by default in MXML.

Table 2.1: List of XES extensions and the attributes they define.

Extension	**Key**	**Type**	**Attribute Level**	**Description**
Concept	name	string	log, trace, event	Generally understood name.
Concept	instance	string	event	Identifier of the activity whose execution generated the event.
Lifecycle	model	string	log	The transactional model used for the lifecycle transition for all events in the log.
Lifecycle	transition	string	event	The lifecycle transition represented by each event.
Organizational	resource	string	event	The name, or identifier, of the resource having triggered the event.
Organizational	role	string	event	The role of the resource having triggered the event, within the organizational structure.
Organizational	group	string	event	The group within the organizational structure, of which the resource having triggered the event is a member.
Time	timestamp	date	event	The date and time, at which the event has occurred.
Semantic	modelReference	string	log, trace, event, meta	Reference to model concepts in an ontology.

For example the events 'create (start)' has a different identity than the events 'create (complete)' and 'submit (start)' according to this classifier definition. Another example of a classifier is a combination of the event name and the resource name. Multiple classifiers can be defined in the log and each classifier has a unique name for identification.

Since no specific set of attributes is specified for an XES event log, the semantics of the attributes these elements do contain are ambiguous. This ambiguity can be resolved by the concept of *extensions* in XES. An extension defines a set of attributes on any level of the XES log hierarchy and in doing so provides points of reference for interpreting these attributes. In the XES event log the extension definitions of the extensions used are referred to in the log tag. Each extension has a prefix at the beginning of the attributes key (e.g. ext:key). The extension definition defines what attribute keys for each element in the event log belong to the specified extension. By explicitly specifying the attributes of elements by the use of extensions a semantic meaning can be attached to the value of those attributes. It should be noted that it is always possible to add attributes to elements which are not defined in extensions.

XES comes with 5 standard extensions to cover recurring requirements on information stored in event logs. The attributes defined by each of these standard extensions are listed in Table 2.1 which is taken from [15]. The *concept* extension specifies a *name* for the log, trace and event elements. The name of a log element might include information about the process this log describes. The name of a trace describes what specific instance is recorded so an identifier is most likely included. For events the name represents the name of the activity that the event represents. The *concept* extension also defines an *instance* attribute for events. This attribute represents an identifier of the activity instance whose execution has generated the event. The use is similar to the use of the *name* attribute for traces. It allows an event in the event log to be connected to a specific record in the data source.

Event types are provided by the *lifecycle* extension in the *transition* attribute for events. The

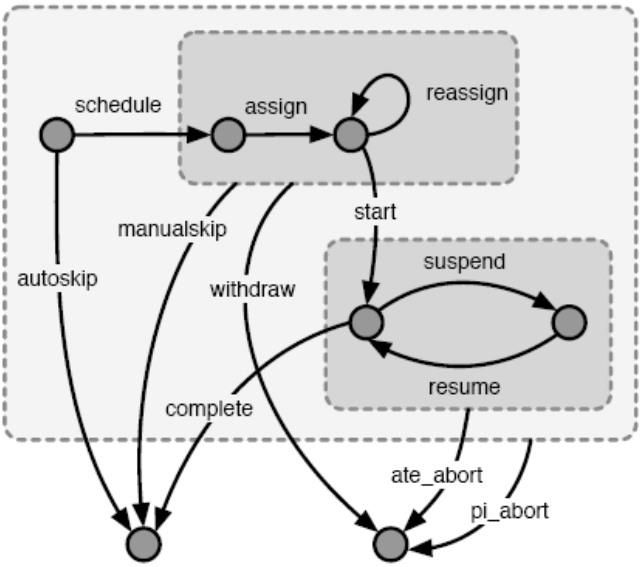


Figure 2.4: Transactional model of the event lifecycle.

standard transitional model is shown in 2.4. It shows the different states an activity can be in. Two of the most common event types in event logs are the *start* and *complete* types. These indicate the start respectively the completion of a certain activity. If there is no information about both the start and end time of an activity only the *complete* event type is used. This extension also defines a *model* attribute for the log element to indicate which transactional model is used.

The third standard extension is the *organizational* extension. This extension defines three different attributes for events. The most important attribute is the *resource* attribute which records the name or identifier of the actor that performed the event. Additionally the role of the resource can be recorded in the *role* attribute. One can also record the group the user belongs to in the *group* attribute.

The next extension is the *time* extension. This extension specifies a *timestamp* attribute for events. This attribute simply records the date and time the event occurred.

The last extension in the list is the *semantic* extension. This extension adds a *modelReference* attribute to all the elements in the event log. This attribute then refers to a model concept in an (external) ontology. Events can for instance contain a reference to the activity they belong to. In the ontology these activities might be stored in a tree structure defining activities on different levels of detail. By using this ontology activities can be grouped together to reduce the level of detail. Another example is including a reference to the resource in an ontology which describes the hierarchy within the company.

Together these extensions define a superset of the attributes used within MXML. The semantic extension is included in the SA-MXML standard [4]. MXML event logs can therefore be converted to XES by using these standard extensions. Data attributes within MXML can be added as normal XES attributes. Attributes of XES extensions that are not present in the MXML event log can be left empty in the XES event log. It is also possible to include default values for missing attribute values. XES event logs can also be converted to MXML if desired. Please note that classifiers and globals are not present in MXML and will be lost in the conversion.

The XES standard is implemented in the OpenXES Java library[2] as a reference implementation. This library provides an interface to read and write XES event logs. The OpenXES library is also capable of (de)serializing XES logs to or from MXML logs.

[2]See `http://code.deckfour.org/xes`

2.3 Process Mining and ProM

Process mining is the art of extracting non-trivial and useful information about processes from event logs. As can be seen in Figure 2.5, process mining closes the process modeling loop by allowing the discovery, analysis and extension of (process) models from event logs. One aspect of process mining is control-flow discovery, i.e., automatically constructing a process model which describes the causal dependencies between activities [3]. The basic idea of control-flow discovery is very simple: given an event log containing a set of traces, automatically construct a suitable process model "describing the behavior" seen in the log. Another aspect is the organizational aspect where the focus is on who performed which actions [10]. This can give insights in the roles and organizational units or in the relations between individual performers (i.e. a social network).

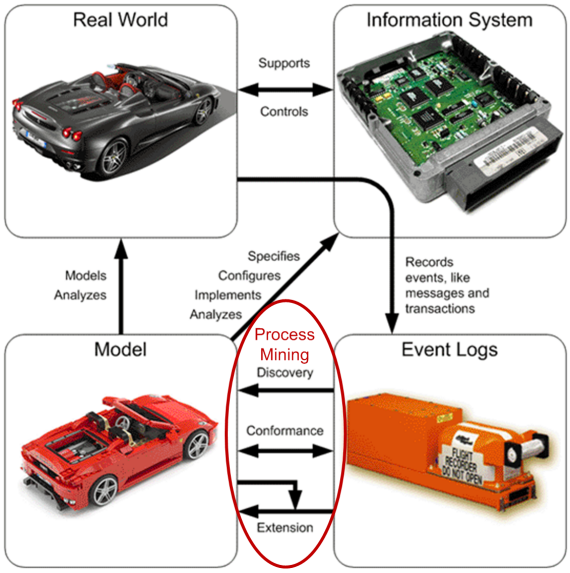


Figure 2.5: From event logs to models via process mining

ProM is a generic open-source framework where various process mining algorithms have been implemented [1] (more than 280 in ProM 5.2). The framework provides researchers an extensive base to implement new algorithms in the form of plug-ins. The framework provides easy to use user interface functionality, a variety of model type implementations (e.g. Petri nets, EPCs) and common functionality like reading and writing files [14]. Each plug-in can be applied to any (compatible) object in the common object pool allowing the chaining of plug-ins to come to the desired result. In most cases the starting input is an event log. ProM can read event logs stored in the formats MXML and from Version 6[3] also in the new event log format XES (as explained in Section 2.2.2). Furthermore, it can also load process model definitions in a wide variety of formats.

To exemplify the use of ProM Figure 2.6 shows the result of the Fuzzy Miner [17] applied on an event log. The fuzzy miner extracts a process model from an event log with the special property that it hides or clusters less frequent behavior. This is especially useful for unstructured processes where other process mining algorithms might produce 'spaghetti' like models. Figure 2.6 shows these clusters as the green hexagon elements. It is possible to look into these clusters to examine the contents. The yellow squares represent events which are connected to other events or clusters by arrows. On the right hand side several settings can be changed. Examples are filters for when to cluster parts of the process and when to show relationships between events. The ProM framework includes many more plug-ins that can reconstruct process models.

Another example of analysis and way of visualization within the ProM framework is shown in Figure 2.7 which shows the result of the dotted chart analysis plug-in. In the example shown

[3]To be released, unofficial nightly builds are provided via `http://www.processmining.org`

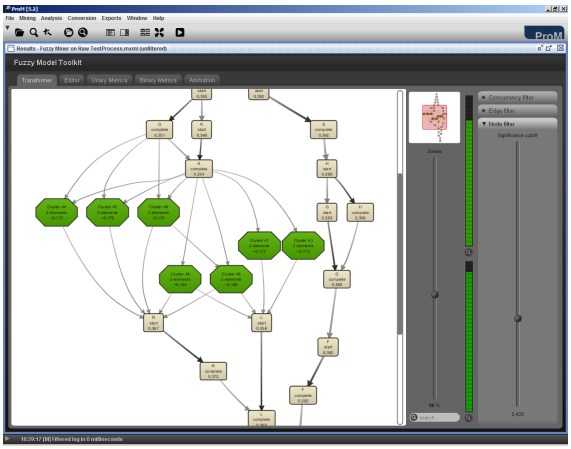


Figure 2.6: ProM 5.2 Fuzzy Miner result.

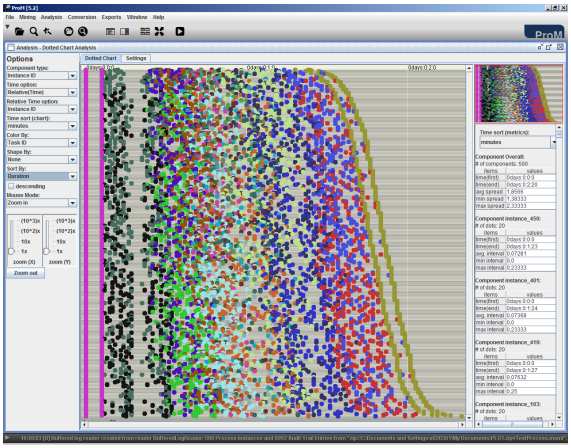


Figure 2.7: ProM 5.2 Dotted Chart result.

each row represents a single case and each dot represents the start or end of an activity on that case. The horizontal axis is set to relative time and cases are sorted based on their total duration. This particular combination of settings is very useful to detect the time distribution between events and cases. As can be seen on the far left of the diagram the pink dots all have the same distance between them which indicates a very constant time between start and completion of the same activity. Another observation that can be made is that all cases end with the same color dots and that the distribution is quite even. Using the dotted chart analysis plug-in many more representations of the log can be made each focussing on different aspects.

Please note that Figures 2.6 and 2.7 show only two out of more than 280 plug-ins. It is impossible to give a complete overview of Prom in this thesis. More information about process mining in general, the most recent research in this area and about the tool ProM can be found at `http://www.processmining.org`.

2.4 Other conversion tools

This section will evaluate two different tools that could be used to fulfill our goal. They both allow converting data from a data source to an event log format. Section 2.4.1 discusses the ProM Import framework which was developed for converting data sources to the MXML format. Another tool build specifically for the extraction of event chains from an SAP database is the

EVS ModelBuilder SAP adapter which is discussed in Section 2.4.2. Besides these two specific applications to convert data to event logs, data conversion is a well studied problem. Generic data conversion will be discussed briefly in section 2.4.3.

2.4.1 ProM Import Framework

To support the recurring task of converting data sources from different systems to the MXML format the ProM Import Framework is created [14, 16]. The main purpose of the framework is to support the process miner in defining and executing the data conversion by providing a programming framework with supporting functions and an easy to use user interface. The main focus point of the framework is extensibility by the use of plug-ins for specific source formats while still providing common functionality from the framework. Another important aspect was the efficiency of writing the MXML file while events might not be retrieved from the source in the order as they should be written. The ProM Import framework has enabled the use of real event logs for testing new algorithms and has also enabled process owners to analyze their process executions.

Although the ProM Import framework has proven very useful in converting logs from real systems to the MXML format it also has some limitations. By allowing extensibility by the use of plug-ins written in Java, which use the framework functionality, new conversions can only be created by people who know the framework's structure and who know how to program in Java. Since not every process miner is also a programmer, not every user can create a new conversion plug-in. Especially people from business such as business analysts with knowledge of the source system find this difficult. A second assumption made during the development of the framework was that the source format would be some form of an event log. The case that the data is spread over a collection of interrelated tables or data sources was not taken into account. As a result of the first assumption, the conversion logic of a plug-in is hidden within the code and there is no intuitive way to define the conversions. Conversions are not always straightforward. As a result of these assumptions the analysis and discussion of the conversion logic with domain experts is rather difficult and can not be done from the conversion code itself.

2.4.2 EVS ModelBuilder

The Enterprise Visualization Suite (EVS)[4] allows for applying a combination of process mining, data mining and statistical functions on event chains [18]. Event chains are a more generic interpretation of traces. The events in an event chain do not necessarily relate to a single process instance. The main reason for not relating events to a single process instance is because of the complexity of information systems such as SAP. In these systems there is not always a clear mapping between events and process instances. By not requiring events to be related to process instances event chain extraction is made easier.

One of the focus points of EVS is allowing for real-time analysis which is facilitated by using an integrated search engine to search through and select matching event chains. The application can use ontologies to structure events, originators and data attributes. Input formats at the moment are MXML event logs, Excel files (in a certain format) and a SAP database using a mapping defined in the 'EVS ModelBuilder'.

The EVS ModelBuilder allows a user to define a mapping on a SAP database in order to extract event chains [19]. An example of such a mapping is shown in Figure 2.8. The grey-brown rounded rectangular shapes represent business objects. Within SAP business objects can be seen as static instances such as orders and invoices but also users, vendors and departments. In the example shown these business objects are a purchase order (PO) and purchase request (PR). Recall Section 2.1.1 where the purchase order is stored in the orders table *EKKO*.

Events are defined by the bottom two shapes. Events have basic attributes such as a date and time of execution. Events are related to business objects. Relations can be of the type consumption

[4] See `http://www.businesscape.no/products.html`

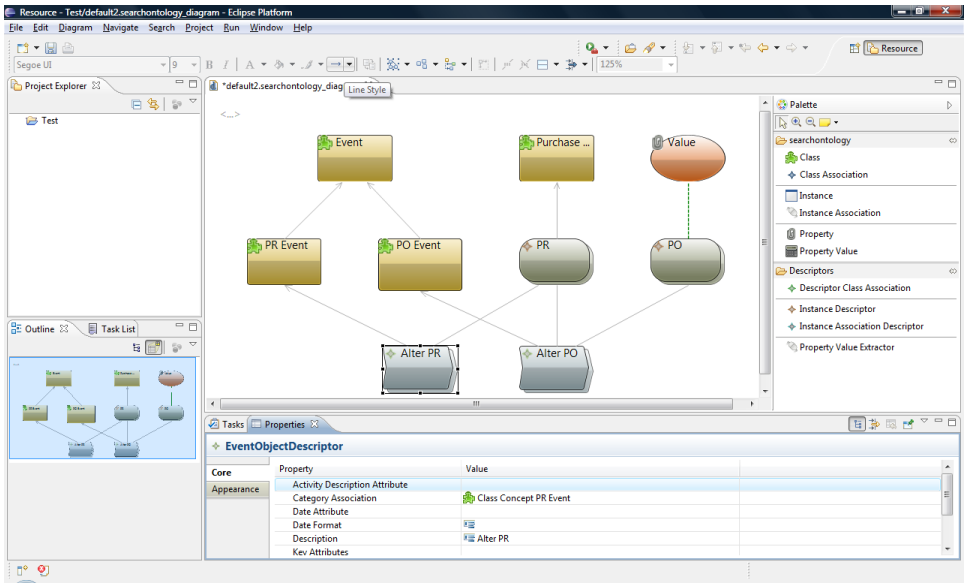


Figure 2.8: Screenshot of the EVS ModelBuilder

and/or production describing whether the event uses and/or produces (or changed) the business object. In this example the events are the change events for both business objects, hence they are related to the business object that is changed. The relationship types for both relations are input as well as output since an existing object is altered. Both business objects and events can have additional properties defined. In the example the purchase order object has a property 'value'. Furthermore, business objects and events can be related to classes. Classes can also be related to other classes hence creating an hierarchical ontology. The 'alter PR' event in the example belongs to the 'PR event' class. Similarly the 'alter PO' event belongs to the 'PO event' class. These two classes are subclasses of the 'event' class. This enables grouping of events and/or business objects to aggregate items during analysis.

The result of the mapping definition is an event chain. Since an event chain doesn't relate to a single process instance the chain is started from the last event and traversed back through relations between events and business objects. For both events and business objects attributes define the SAP document number and a unique identification key. Using the relations between events and business objects the two can be related in the data source. This relation can be either explicit which means that there is a direct relation from source to target using only one table in the database. Implicit relations specify a path from the source table to the target table to be able to connect the items to each other.

Although the EVS ModelBuilder is currently specifically aimed at extracting event chains from an SAP database the same principle can be applied to any data source that is based on database storage. One of the complicating factors for using the EVS ModelBuilder to extract MXML or XES event logs from the data source is the absence of a relation between events and a single process instance. Furthermore, each event needs to be defined explicitly. Also, the ModelBuilder details were only available near the end of the project discussed in this thesis. All these reasons together support the decision that the EVS ModelBuilder can not be considered a valid solution to our project goal. Lessons can however be learned from the approach taken by EVS ModelBuilder. These are discussed in Chapter 7.

2.4.3 Generic Data Conversion Solutions

Converting data from one source to another and the definition of a translation is a very common problem. In [12], for instance, the Clio project is discussed in which data integration and

data exchange between different data structures are the key topics. Another domain where data conversion is a common problem is in data warehouses. Information needs to be extracted from many systems and stored in a single data source [25]. Unfortunately, the solutions created within these domains can not be used for our problem. The main problem is that these tools are very generic and not specific for a certain output format. To provide more guidance to the user a more specific application needs to be developed to convert data to an event log.

Chapter 3

Conversion Aspects

As discussed in the previous chapters, there is a need for a tool to extract event logs from a data source. But before such a tool can be developed, the characteristics of the conversion should be investigated in detail. In general the conversion consists of two steps. The first step is the definition of the conversion. This conversion definition specifies how concepts of the data source are mapped onto concepts of the event log. The second step is to execute this conversion and actually convert the data from the data source to the event log as specified in the mapping. It is important to be aware of the influence of decisions made in the mapping phase on the resulting event log and hence the process mining results. This section discusses several key aspects that are important in defining the conversion.

Defining a conversion is often not trivial and several choices should be made. To complicate things even further there is no single correct conversion, the conversion definition depends on the desired view on the data. Even within the same process mining project there may be a desire to view the process from different angles. This can result in defining multiple conversions to extract different event logs from the same data source. Furthermore, errors made in the conversion definition can lead to serious problems in later steps. All in all the event log extraction is a very important step in the process mining process which, up until now, is often neglected.

As discussed in Section 2.2 an event log generally consists of event traces. Each trace records the activities that were executed on a certain process instance or case. Traces and events, as well as the log as a whole, can contain several attributes recording information relevant for analysis. Events record information about the execution of tasks on cases. This information is stored in attributes of the event. Information such as the name of the activity, when the task started or stopped and who executed it are common event attributes.

One of the first steps in a process mining project is to determine the goal of the project which is further discussed in Section 3.1. The first conversion choice often is to select the process instance of the process. Different aspects influencing this decision are discussed in Section 3.2. Once the trace representation is determined, the event selection can start as discussed in Section 3.3. There is a problem however if traces and events are not uniquely related as is discussed in Section 3.4. This is followed by the selection of what attributes to include, which is discussed in Section 3.5. The last section, Section 3.6, discusses ways to limit the number of traces and events included in the event log.

3.1 Project Decisions

The definition of the conversion from the source data to XES greatly depends on the **goal** of the process mining project. Therefore the goal of the process mining project should be clear before the conversion can start. In many cases the executed process is not known and the goal of process mining therefore is to visualize the executed process as recorded in the event log. Another goal of a process mining project can for example be to look at certain performance aspects of the process

and their possible causes. Verification of rules such as the 4-eyes principle is another example of a process mining goal. However, in many cases there is no clear definition of the analysis that should be conducted. In these cases a more general explorative analysis of the event log on different aspects is done as a first step [6]. The findings of such an explorative analysis can trigger other process mining projects to examine certain details of the process further.

As a result of the goal of the project the **scope** can be determined. This is important for the conversion because it determines what should or shouldn't be included in the event log extraction. The scope is partly determined by the goal of the project. The goal often hints what part of the overall process should be investigated. Consider for instance that the goal of the project is to examine the procurement process. The scope of the project will most likely be limited to events related to procurement orders. The part of the process related to goods movements within the internal warehouse will be out of scope.

Another resulting factor is the **focus** of the project. If the goal of the project focusses on certain parts of the process then extra detail should be included for these parts in the event log. As will be discussed later, this can result in more fine-grained events to be included in the part that has more focus. Additional attributes for more detailed analysis can also be included. In the example of the procurement process investigation, focus could be more on the order payments than on the time it takes for delivery.

3.2 Trace Selection

As discussed in Section 2.2, an event log contains *traces* of events. In general each trace relates to a certain process instance or case in the system. Hence, each trace contains events that occurred for that process instance. See also Figure 2.2.

Candidate process instances are often recognized business objects. Business objects are objects or items handled or used by the business. Examples include patients, machines, robots, orders, invoices and insurance claims. Sometimes trace candidates can also be less tangible objects such as treatments in a hospital, machine usage between start up and shut down, user sessions on a web site, e-mail conversations etc. In general, business objects are stored in so called master data tables in databases [19]. These master data tables record information about business objects and can therefore be used to add certain attributes to the trace as will be discussed in Section 3.5.

It is important to note that in one event log there is only one type of process instance to which the events are related. For instance it can not be the case that certain events in the event log only relate to a patient while other events only relate to a machine such as maintenance activities. Therefore, a trace should contain events related to a single business object. This common business object type is our process instance.

The selection of the trace determines or is determined by the scope of the project. By selecting a certain business object as the trace the scope is set. Only events related to the trace object can be included in the event log. Consider for example a hospital which treats patients following certain treatment procedures. If one patient is considered to be the process instance one can investigate which treatments patients undergo including possible relationships between different treatments. However, if a treatment itself is taken to be the process instance, as could be defined by the scope of the project, no information about the overall process the patient has gone through is available. In the latter case only separate treatment executions can be investigated and therefore the project scope is smaller than when a patient is considered to be the process instance. One could even select the process instance to be a machine used in several different treatments such as an X-ray machine. These machines often keep detailed records of their usage [24]. When selecting such a machine as a process instance one looses detailed information about the treatment process and the process the patient went through. Another example is a procurement process which includes multiple business objects to follow. Possible candidates are order requests, orders, order lines, receipt of goods or invoices. Each one is in theory a valid choice and certain events could be related to all these objects. Nonetheless each object has a different set of related events. Hence, process mining results will differ depending on the trace selection. Therefore, the selection of the

process instance determines the scope of the process investigated.

3.3 Event Selection

After the selection of the process instance to which events should relate, the selection and specification of these events can begin. Which events to include is determined by the focus of the process mining project. The focus of the project can be on a certain part of the process. This means that preferably (more detailed) events of this part should be included in the event log. While events of other parts of the process could be omitted. It is important, however, to explicitly choose the level of detail of events. In general the entire event log should be at a consistent level of detail. During process mining all events are treated equal. A uniform level of detail is therefore important for correct conclusions to be drawn in the analysis phase. However, if one knows that the level of detail of certain events is different than that of others this can be taken into account during the analysis phase to prevent wrong conclusions. Care should be taken that events are not defined at multiple levels of detail. For instance if the activity 'create order' consists of two more detailed activities. Either the 'create order' activity or the more detailed activities should be included. Otherwise it might appear from the event log that 2 activities are performed in parallel on a single case. In reality however, there is only one activity performed on the case but this is recorded 2 times.

The ideal solution to the problem of the detail of events is to create an ontology of activities to which each event refers. This ontology allows the grouping of activities to form a higher level activity. For example the activity *Pay invoice* can consist of sub-activities such as *Check invoice amount*, *Check if goods are received* and *Prepare payment*. By recording the lower level sub-activities in the event log these activities can be grouped together as the *Pay invoice* activity during the process mining phase by using the ontology. More information about using semantic ontologies in process mining can be found in [4], [5, Section 3]and an example application in [18].

Another aspect in the level of detail of events is the type of events recorded in the event log. Since events are atomic recordings of executed activities they don't have a duration. To record the different states an activity can be in, each event can be of a certain type. The transactional model of event types used by ProM is shown in Figure 2.4 (page 15) as a state machine. The most commonly used event types are *start* and *complete* to indicate the start and completion of a certain activity. It is possible to record only events of one type (most commonly the *complete* type is used). Adding the *start* type may be mandatory for certain types of analysis, e.g. for measuring actual processing times. Especially for performance measurements the event type is important. The event types *start* and *complete* are used to distinguish between waiting time and processing time. Waiting time, or queueing time, is the time between the completion of one activity and the start of the next for a certain case. Processing time is the time required to perform a certain activity. Therefore, adding these event types can be very useful in the analysis of the event log and enable new analysis methods.

One of the other aspects of event selection is the choice of defining each event separately or reusing some kind of event log, or both. The latter option might sound strange but in many systems there usually is some form of log present. Often the format of the log is not directly suitable for process mining. For example because some events do not relate to a single process instance. It can also be the case that certain information is missing. These logs however do provide valuable information on what activities are executed. One can either directly extract a part of this log, enrich it with missing data and convert it to a more suitable event log. Another option is to examine the log to get an idea of the activities in the system. One can then specify each activity separately. The downside of specifying each event separately is that possibly not all events of the system are included. Events can be overlooked or just be unknown and therefore do not appear in the event log. Timestamps in the data source provide good hints for event recordings. The advantage of specifying each event separately is that the level of detail can be kept consistent. In reality the event definition is often driven by the information present in the data source. This then results in the definition per event, each with its own characteristics.

Table 3.1: Example showing both convergence and divergence between the order trace and payment events.

Order		
ID	**Value**	**Created On**
1	€ 100	01-01-2010 09:00
2	€ 100	13-01-2010 15:47
3	€ 100	05-02-2010 17:01
4	€ 100	13-03-2010 08:45

Payment		
ID	**Amount**	**Payed On**
10	€ 150	31-01-2010 18:00
11	€ 25	03-02-2010 11:12
12	€ 25	03-02-2010 15:14
13	€ 75	25-02-2010 12:55
14	€ 25	28-02-2010 16:03
15	€ 100	30-03-2010 08:46

Order × Payment	
Order ID	**Payment ID**
1	10
1	11
1	12
2	10
3	13
3	14
4	15

3.4 Convergence and 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. This is not always the case in reality. Hence it is a problem in the conversion to event logs. This section discusses the problem in more detail. Implications for the selection of traces and events and for the analysis phase are also discussed.

Table 3.1 provides an example demonstrating the problem to be discussed. The example data consists of orders stored in the *order* table. Each order has a unique identifier stored in the *ID* column. Furthermore, each order has a total amount that needs to be payed as stored in the *Amount* column. Payments are recorded in the *Payment* table. Each payment also has a unique identifier which is stored in the *ID* column. The amount payed in each payment is stored in the *Value* column. To connect orders to payments and vice versa the *Order* × *Payment* table records these relationships. However, one payment does not necessarily pay for a whole order at once, it might pay only a fraction. For instance payments 13 and 14 together pay for order number 3 as can be seen in table *Order* × *Payment*. To complicate things even further, a payment might correspond to (parts of) two orders. In this example payment 10 pays for both orders 1 and 2. In this particular case we could derive that 100 of the 150 Euros payed is payed for order 2 while the remaining 50 Euros is payed for order 1. In general, it can not be decided how the payment is divided over the orders it corresponds to.

In this example the order is chosen as our process instance. Payments are the events that can occur for orders. The event log with its traces and events that would be the result of this simple conversion is shown in Figure 3.1. For this example additional data such as the user who performed the payment or the event type is not considered for reasons of simplicity. In practice these attributes would of course be included in the event log. There would probably also be more events in the event log than this single payment event.

The problem of one event relating to multiple process instances is called **convergence**. According to [23, p.42] this is defined as follows:

***Convergence**: The same activity is executed on multiple process instances at once.*

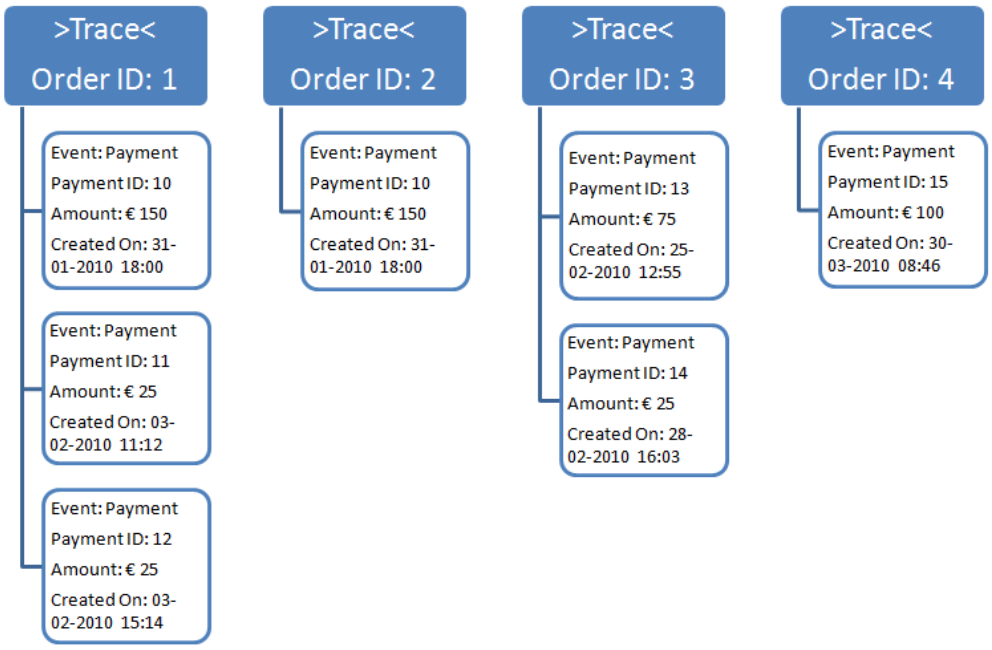


Figure 3.1: Event log result from the data in Table 3.1.

This property can be recognized by a 1:n relation from an event to the process instance in a database structure. The result is that a single event occurs in multiple traces in the event log.

Convergence can also be identified in our SAP example from Section 2.1.1: change events in the *CDHDR* table might relate to several orders or order lines at once.

In the example event log as shown in Figure 3.1 convergence also occurs. The payment with ID number 10 appears in the traces from both orders 1 and 2. Note that all characteristics of payment 10 for both orders are exactly the same. Hence, during process mining analysis it would appear that a certain user was executing two payment activities at once. One can imagine the influence this can have when it occurs on a larger scale in event logs. The utilization of resources would not be reliable any more. Besides influencing the utilization of resources 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.

The same problem can occur the other way round and is then called **divergence**. According to [23, p. 41] this is defined as follows:

***Divergence**: For one process instance the same activity is performed multiple times.*

This property can be recognized by a n:1 relation from events to the process instance in a database structure.

In the example event log of Figure 3.1 this property occurs for instance for order 3 which is payed by payments 13 and 14. It also occurs for order 1 where a part of payment 10 together with payments 11 and 12 pay the total amount. The effect on process mining magnifies itself in the resulting process model. Process model discovery algorithms specify each activity only once. If an event occurs multiple times within a trace the algorithm will try to reuse the same event instance multiple times. If no other events occur between multiple executions of the same activity this results in loops in the process model. Not all process discovery algorithms can handle event loops. However, if other events do occur in between the process model will become more complex.

A result of convergence and/or divergence can also be that certain process mining algorithms are unable to handle the event log. The Alpha miner for instance is not capable of mining event logs where convergence and divergence occurs [23].

To solve convergence and/or divergence one could change the representation of the process instance in the conversion. In the example provided one could for instance change the process instance from the order to the order line. This could solve the problem of divergence if each payment would only relate to one order line. The problem of convergence would still remain since payment 10 will always pay at least 2 order lines, one of order 1 and one of order 2. Hence, convergence and divergence can not always be solved. *Convergence and divergence should however always be taken into account during the process mining phase.*

It should be noted that convergence can be recognized by process mining algorithms if the *instance* attribute of the *concept* extension of XES is provided. By comparing events using the activity instance identifier these events could be recognized and treated differently. Unfortunately, none of the current algorithms uses this ability.

3.5 Attribute Selection

The log and its traces and events contain attributes to record certain information. Attributes can even contain attributes themselves to store more detailed information. Selecting which attributes to include is not a trivial task. If too many attributes are included, the event log becomes unnecessarily large. This makes it hard to handle, load into various tools and apply algorithms on it. If too few attributes are included, this limits the number of different analyses that can be done on the event log. Carefully selecting which attributes to include and which not is therefore very important.

The first type of attributes discussed are those for the log in Section 3.5.1. For traces and events two types of attributes are defined. The first type are generic attributes which are discussed in Section 3.5.2. Data attributes are discussed in Section 3.5.3.

3.5.1 Log attributes

The event log contains a log element as its root which contains all the traces. This log element can also contain attributes. Since it only occurs one time in the event log the impact of including many attributes is small. It is however important to include relevant information describing the contents of the event log and its origin. The following attributes should be considered to be added to the log element:

Process Name A name of the process for which this event log records the execution.
Example: "Procurement process" or "Neurology treatment process".

Data Source A description of the information system from which the event log is extracted.
Example: "SAP ERP system" or "X-Ray machine log".

Source Organization The name of the organization providing the data.
Example: "General Hospital of X" or "Company X, Division Y, Location Z".

Description A brief general description of the contents of the log.
Example: "Procurement process with order as trace object for performance analysis with all orders created in 2009."

Version To identify different versions of event logs.
Example: "2010-01-28 10:13:22" or "v1.01 (corrected for order amount attribute)".

Author Name and contact details of the one who defined/executed the conversion.
Example: "Joos Buijs (j.c.a.m.buijs@student.tue.nl)"

Process Mining Project A reference to the process mining project or the purpose of the event log.
Example: "Process discovery of the procurement process within company X."

Additionally one should consider the attributes defined for the log element by the five standard extensions of XES (see Section 2.2.2). The name of the process as defined by the *concept* extension is already mentioned. Another attribute for the log element is the *model* attribute defined by the *lifecycle* extension. This attribute specifies which lifecycle model is used for typing events. The last attribute defined by one of the five standard extensions is the *modelReference* attribute as defined by the *semantic* extension. The *modelReference* attribute refers to model concepts in an ontology. For the log element it can refer to the process definition and the level of abstraction of the recorded process.

3.5.2 General Attributes for Traces and Events

The attributes defined for trace and event elements are repeated and thus may appear frequently in the event log and should therefore be carefully selected. In general, two types of attributes for traces and events can be defined. The first type are those attributes that can be specified for traces and events in general. Some of these attributes are more or less required by process mining algorithms. Most of these attributes are defined by the five standard extensions of XES. The second type of attributes are data attributes which are discussed in the next section. Data attributes store additional information about the object the trace refers to or the activity executed.

Certain algorithms require specific attributes to be present in the log. However, not all attributes defined by the standard extensions should always be present in the event log. Sometimes the information required is not even available in the data source.

The most important extension is the *concept* extension which specifies a *name* for the log, trace and event elements. Providing names for each element is easy and very informative and should therefore always be provided. Names of traces should include a unique identifier. This enables later investigation of specific process instances if interesting observations are made. Names of events should provide the name of the executed activity represented by the event. The *concept* extension also defines an *instance* attribute for events. This attribute represents an identifier of the activity instance whose execution has generated the event. Including this attribute in the event log allows for matching events to data records in the source system to enable further analysis. It could also help in the detection and correct handling of convergence as discussed in the previous section.

Another important extension is the *time* extension. This extension specifies a *timestamp* attribute for events. By recording the date and time the event occurred in a timestamp events can be ordered. Furthermore, this enables performance and duration analysis. It is important to provide as much detail in the timestamp as possible. Preferably up to the level of seconds and sometimes even milliseconds. Especially in machines many events can be executed within the same second hence requiring precision up to the millisecond. This is necessary because it is very important that events have unique timestamps within a trace. Although most algorithms can handle equal timestamps results are drastically improved when more precision is provided.

Now that elements have names and events can have timestamps it is important to acknowledge that events can be of different types. As discussed before events are atomic recordings and hence have no duration. Since activities do have a duration events can be of different types each recording a different state of the activity. Event types are provided by the *lifecycle* extension. Two of the most common event types are the *start* and *complete* types. These indicate the start respectively the completion of a certain activity. Providing these two event types in the log for each activity allows for estimating the processing times and waiting times in performance analysis. This is a very important distinction in analyzing the performance of a process. More event types exist as shown in Figure 2.4. In most cases *start* and *complete* suffice. If there is no information about both the start and end time of an activity only the *complete* event type is used. Remember that the transactional model used should be declared in the log element with the *model* attribute.

Another analysis dimension in process mining is that of work distribution. By relating events to resources, or groups of resources, the distribution of the work over different units can be visualized. Furthermore, social networks can be constructed to indicate which actors worked together on cases, handing over work, etc. One could also detect groups of actors performing similar tasks etc. To

enable the discovery of these networks, the actor or group that executed the event should be stored with each event. This is defined by the *organizational* extension. This extension defines three different attributes for events. The most commonly used attribute is the *resource* attribute which records the name or identifier of the actor that performed the event. Additionally the role of the resource can be recorded in the *role* attribute. One can also record the group the user belongs to in the *group* attribute. Which of these attributes to use depends on the information available and the level of detail desired. In some cases analysis on the group or role level is desired. For other types of analysis, such as the 4-eyes principle, the user that executed the event should be recorded.

The last of the five standard extensions is the *semantic* extension. This extension adds a *modelReference* attribute to all the elements in the event log. This attribute then refers to a model concept in an (external) ontology. This can be applied to events allowing them to be grouped. As discussed in Section 3.3 this allows for changing the level of detail in the process model by expanding or collapsing event nodes. Another example is in using it to refer to an ontology of users in the resource attribute of the organizational extension. One could also define an ontology of resources grouping them by role and group at once. This would make the role and group attributes of the organizational extension redundant.

Other extensions and attributes might exist that define process related attributes. The five standard extensions discussed above however capture the most frequently used attributes. At least a subset of the attributes defined by these extensions should be specified in the event log for process mining to be successfully applied.

3.5.3 Data Attributes for Traces and Events

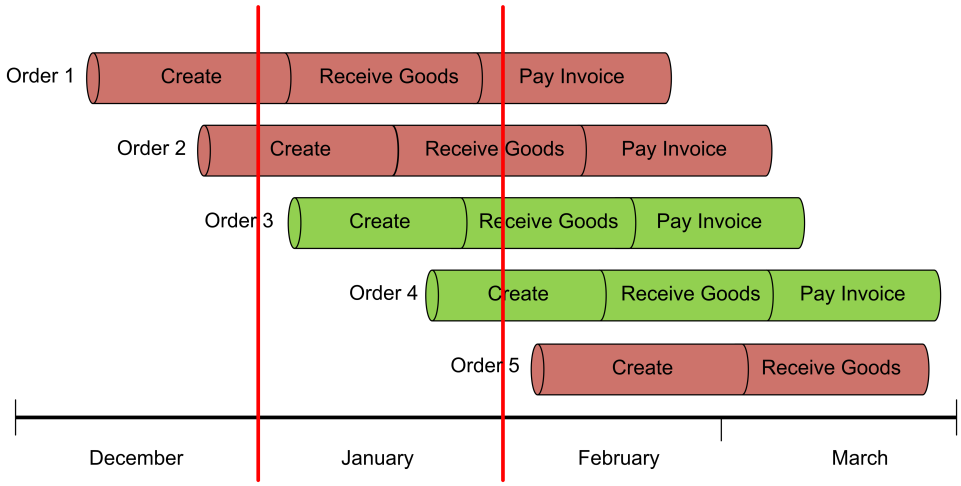
The next type of attributes are data attributes. These attributes store properties of the process instance or activity executed. Since process instances and activities are different within each event log there are no attributes specified beforehand. Data attributes might even be absent in the whole log and process mining can still be performed. Data attributes can however provide important information about the process. For example data mining techniques can be applied to discover the business rules. These rules are for instance used to decide whether or not a certain activity should be performed in the process. One could also use data attributes to discover which properties of process instances indicate beforehand that the execution takes longer or that a certain group or originator will be involved. Data attributes can also be used to check certain rules. One could for instance require that payments above a certain amount can only be executed by managers. Please note that the attributes related to process mining such as timestamp and resource can also be used in data analysis.

Which data attributes to include for traces and events is not trivial. Often it is not known beforehand which properties are used in business rules. If the rules that will be checked for are known beforehand, then it is clear which attributes to include. In general process instance attributes are very useful. Examples of the most common attributes are often regular dimensions such as the order value, patient age or weight of the object. Useful event attributes are most often the values of properties of the process instance before and/or after the event occurred.

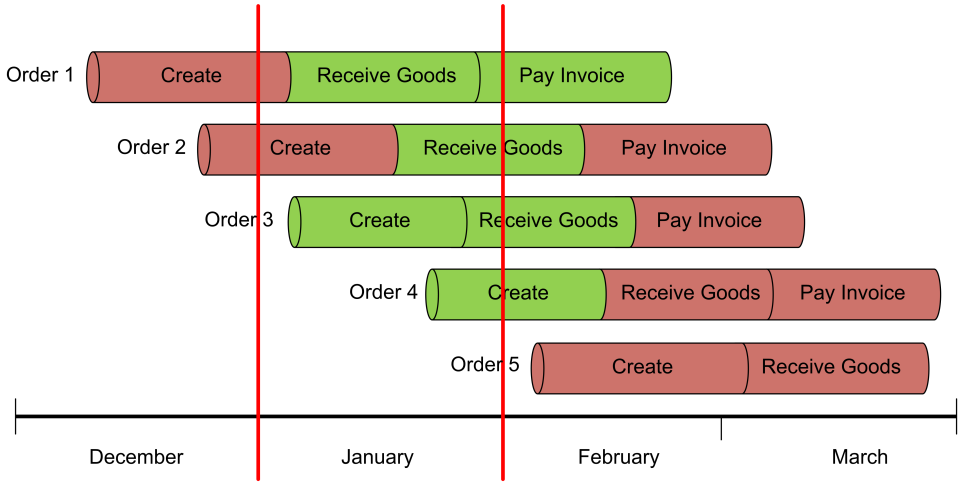
3.6 Filtering Traces and Events

Besides selecting which attributes are included in the event log one can also select which traces or events are included in the event log. In general there are two ways of achieving this: (i) by selecting only a limited number of traces or (ii) by only selecting a limited number of events.In this section we will show filtering on time for both traces and events. One could also filter on traces (not) starting or ending with certain events or having certain other properties.

Figure 3.2 compares these two filtering methods using an example. In these examples time plays a role in the selection. In the figure there are five traces identified in the source system each



(a) Limiting on the trace level



(b) Limiting on the event level

Figure 3.2: Comparison of trace limitation and event limitation.

referring to an order. Events in this example are the creation of the order, receipt of the goods and payment of the invoice.

The most commonly used method is to select a number of traces to include in the event log. This is shown in Figure 3.2a. In this example all orders created in January are included. Orders 3 and 4 are therefore included in the event log. In total 6 events are included in the event log. Another example of selection on the trace level would be to select those orders with an identification number within a certain range. The benefit of limiting on the trace level is that if the chosen time period is far enough in history all process instances are finished by now. This means that each trace describes a complete process execution from beginning to end. If the current time would be early February for instance then both orders are not paid yet so the trace of the process is not complete. However if we are now in April then both selected traces would describe the complete process.

Selection on the trace level is therefore preferred when the process model needs to be discovered or if analysis is done on a process instance level. Examples are rule verification and the dotted chart visualization. The downside of this method is that it does not include all events that were

executed within a certain time frame. Think for instance of the receipt of the goods for orders 1 and 2 which also occurred in January. These events are not included in the event log. Therefore it might appear from the log that no goods where received at all in January. Furthermore, from the resulting log it appears that 'create' events only occur in January. However, in February order number 5 is created. Therefore one can not conclude anything about utilization on the level of resources over time using this selection method.

Another approach is to only look at a certain time period (a week, month or year) of the process execution. This method is shown in Figure 3.2b. Only those events are included in the event log that occurred within a certain time period. In the example all events that occurred in January are included. Using this method again 6 events are included in the event log. However, in this example these events are spread over orders 1, 2, 3 and 4. This method also has its implications on the process mining results. None of the traces of these orders now describe a complete execution. This limits the reconstruction of correct process models describing the whole process. On the other hand one now has a good indication on the utilization of resources during the month January.

A possible solution would be to extend the selected time period. As a rule of thumb one should select the time period to be about five times as long as the duration of an average case. Using this rule of thumb the event log mostly includes complete traces. The longer the selected time period is, the more complete traces are included in the event log. During the process mining analysis one can look at which events occur most often at the start or completion of a trace. Next one can extract only those traces that start or end with one of these events. By using this method one can extract those traces describing a complete process execution with a reasonable certainty. This subset of traces can then be used to discover the process model. Please note that the original log, including incomplete traces, should be used for other analysis methods such as social network analysis.

The disadvantage of this method however is that there might be exceptions to the rule. Some process instances might start or end with a special activity. These traces are now excluded from the event log selection. Hence, these exceptions will not appear in the discovered process model.

In general it is more important to have complete traces. For instance for process discovery, which is a common focus of process mining. If resource utilization is the focus of the process mining project then selection on the level of events is recommended.

Indications on the number of traces or events to include are hard to provide. It depends on many factors such as the number of cases in the system. The number of events to include is also limited by the number of events that can be identified. In general, the more traces included in the event log, the better the analysis results are. However, including many traces in the event log can increase the duration of analysis methods drastically.

3.7 Conclusion

In this chapter several aspects of the conversion definition have been discussed. We discussed the influence of the goal, scope and focus of the process mining project on the conversion definition. Next the selection of traces was discussed. This selection is determined by, or determines, the scope of the project. Once the trace instance is identified, related events should be defined. What events to include is mainly determined by the focus of the project. In an event log each event should relate to a single trace instance. In reality this is not always the case. As we discussed, activities might be performed for multiple traces and one trace can contain a certain activity multiple times. This should be recognized during the conversion definition and taken into account during the analysis phase. Once the traces and events are defined, their attributes need to be selected. On the one hand more attributes provide more insight in the process. On the other hand, adding too many attributes makes the event log unnecessarily large. This chapter is concluded by an example on how to filter the number of traces and/or events to include in the event log. The different aspects of the conversion definition discussed in this chapter need to be taken into account while designing our generic solution. In Chapter 4 this generic conversion approach is discussed in detail.

Chapter 4

Solution Approach

This chapter discusses the approach taken to implement the application described in Chapter 1. Section 4.1 discusses the key functionality the application should provide. In Section 4.2 the domain model used to store the conversion definition is discussed. To improve the understanding of the conversion definition a visualization is made which is discussed in Section 4.3. During the conversion execution the data is temporarily stored in a database which is explained in Section 4.4.

4.1 Functionality

As described in Section 1.3 the goal of the project is to create a prototype facilitating the conversion of event data in some source system to the XES event log format. The prototype should help a business analyst in defining and executing a conversion from a database to the event log format XES. This should be achieved with as little programming by the user as possible. From this we can distinguish two main functions of the application prototype:

- Support the user in **defining a conversion** from a source data structure to the XES event log format.
- **Execute the defined conversion**: the XES event log will be created using the logic as defined in the conversion.

To support these two functions the following key functional and non-functional requirements can be formulated. These requirements are important in achieving the project goal.

The first functional requirement that can be formulated is that of *data source connectivity*. The application should be able to connect to the most commonly used type(s) of data sources. On the other hand, the connection interaction should be general to simplify implementation. Most applications store their data in relational databases. A relational database consists of tables which consist of rows and columns of information. Relationships or references between these tables might exist. Therefore, by supporting this type of data source the application can access the data sources of most information systems. By supporting this input format in most cases no manual intermediary conversions or extractions need to take place to prepare the data. The application will be able to read directly from (a copy of) the original data source as used by the information system. Furthermore, there are many libraries available to connect to relational databases of various brands and types. Therefore nearly any relational database can be used as input for the application. Note that there are libraries available that enable comma-separated-value files and XML files to be accessed as if they were relational databases. Microsoft provides such drivers standard with Windows. Other examples of such drivers are CsvJdbc[1], StelsCSV[2], StelsXML[3]

[1] See `http://csvjdbc.sourceforge.net`
[2] See `http://www.csv-jdbc.com/stels_csv.htm`
[3] See `http://www.csv-jdbc.com/stels_xml_jdbc.htm`

and xlSQL[4] but many more implementations exist. Therefore, by choosing relational database systems as our data source input format we enable access to a wide variety of information system data storages. At the same time implementation is simplified by using general libraries with a common interface to interact with all types of data sources.

As discussed in Section 1.3, the application should be able to *generate event logs* in the XES format, which is the second functional requirement. The main reasons for choosing the XES format are that it is more flexible than MXML and is supported in the upcoming version 6 of the ProM framework. The XES format was extensively discussed in Section 2.2.2.

To convert the data in the data source to the elements in the XES event log format the user needs to *define a conversion*. This conversion should precisely describe how the data from the data source can be converted to the XES format. This means that the conversion should be very specific and detailed such that the application can execute it. However, the conversion also needs to be defined by a user without much programming knowledge. Therefore the conversion definition should be specific and concrete on the one hand but also easy to define on the other hand.

The conversion definition can take two starting points. It can start at the data source and extract the elements required by the XES event log format. The benefit of this approach is that the user probably has more knowledge about the data source than the target event log format. The problem however is that every data source is structured different. Furthermore, by hiding the target event log structure the user is not made aware of the particularities of the format. As is discussed in Chapter 3 the decisions made during the conversion have great impact in the analysis results. Therefore, the other starting point of the conversion, the target XES event log format, is a better approach. Using this approach the user needs to specify where each element of the format is located in the data source. One of the benefits of this approach is that the target structure is known beforehand. Therefore the application can provide more guidance to the user. The other benefit is that the user is made more aware of the possibilities and limitations of the target format. Hence, it can make better decisions on how to convert the data source to the event log format. Therefore the latter approach is taken in guiding the user in defining the conversion definition. The choice for an event log oriented approach has great influence on the way the application is structured. This is very clear in the domain model used to store the conversion as will be discussed in Section 4.2.

Another important aspect is how *user friendly* the application is. The application should be easy to use for people without a deep programming or process mining background. Therefore the amount of programming required should be limited. To avoid programming we use the *Structured Query Language* (SQL). This computer language is developed for working with databases and is the most widely used language for relational databases. Using SQL as the language within our application has many benefits. First of all it is not a programming language but a query language and therefore has a much simpler syntax. Furthermore, the application splits up the elements of a complete SQL query in smaller parts. This further simplifies the creation of the entire conversion definition. Most users that have enough knowledge about the database to define a conversion are also able to work with databases in general. Hence, they will most likely already know and understand SQL. Furthermore, using SQL allows the application to easily convert the conversion definition to an SQL query that can be run on the data source. A possible problem from a user perspective is that there are many versions of SQL around. The SQL used in the application should be correct for the database system used.

4.2 Domain Model

The purpose of the domain model is to store all the information required to convert the data source to an XES event log. As mentioned in the previous section the application takes an event log oriented approach on the conversion. This means that for each element in the resulting event log the conversion defines what the value is or where it can be found in the data source. Therefore the domain model features key elements of the XES event log format. These elements

[4] See `https://xlsql.dev.java.net`

are complemented with additional information required to extract the information from the data source. The conversion is stored in the structure as shown in the class diagram of Figure 4.1.

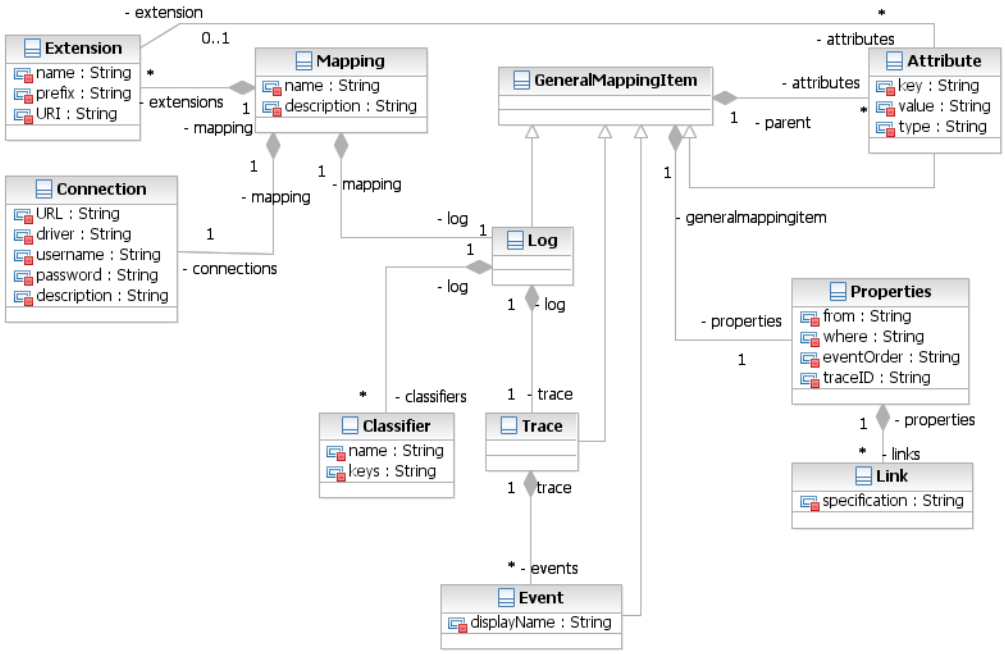


Figure 4.1: Class diagram of the domain model of the application.

Table 4.1 shows part of the data source that we use as a simple example in this section. The data source consists of two tables: 'events.csv' and 'users.csv'. The data source used is actually a set of comma-separated-values files. Each of these files is represented by a table using a special converter. The table name is identical to the file name, including the extension. The 'events.csv' file/table consists of events in an event log format. Each event relates to a unique order number, stored in the 'orderID' field. Furthermore, the name of the event is stored in the 'eventName' field. The date and time of occurrence is stored in the 'timestamp' field. The type of the event is stored in the 'eventType' field. From the user who executed the event only the user identifier is stored in the 'userID' field. The user identifier refers to a record in the 'users.csv' table. This table stores more information about users. Of course, each user also has a 'userID' value for identification. Furthermore, users have a 'userName'. For each user also the group (s)he belongs to and what role (s)he has are stored.

The data of the example in Table 4.1 is used to demonstrate the use of the domain model. Figure 4.2 shows an instance of the domain model of Figure 4.1. The domain model instance defines a conversion of the data from Table 4.1 to an event log.

The root element of the domain model is the `Mapping` element. This class contains attributes for a conversion `name` and a brief `description`. These attributes can be used to describe the specified conversion. The `name` attribute can for example be 'Procurement process in SAP' or 'X-Ray machine repairs in Dutch hospitals in 2009'. The `description` attribute can include a more detailed description of the conversion definition and the decisions made. It can, or actually should, also contain information on who defined the conversion and which version of the conversion it is. In the example of Figure 4.2 the mapping has the name 'Demo Mapping' and the description 'Demo mapping's description'.

To establish a connection to the data source the `Connection` class contains the relevant information. These properties are defined by the protocol used to establish the connection. More on the exact implementation of the connection can be found in Chapter 5. The `URL` property defines the location of the database to connect to, possibly including information such as which

Table 4.1: Selection of events.csv and users.csv table content.

events.csv				
orderID	**eventName**	**timestamp**	**eventType**	**userID**
1	Create	1-1-2009 10:00	Start	1
1	Create	1-1-2009 11:00	Complete	1
2	Create	1-1-2009 11:00	Start	1
2	Create	1-1-2009 12:00	Complete	1
1	Send	2-1-2009 10:00	Start	2
3	Create	2-1-2009 10:01	Start	1
123	Create	14-2-2009 9:00	Start	1
123	Create	14-2-2009 9:00	Complete	1

users.csv			
userID	**userName**	**userGroup**	**userRole**
1	George	Purchase	OrderCreator
2	Ine	Support	Secretary
3	Eric	Warehouse	Reciever
4	Wil	Finance	Payer

port to use and the database name. Drivers exist to enable the connection to different types and brands of relational database systems. In the example of Figure 4.2 the URL is specified as 'jdbc:odbc:CSVMultiple'. This indicates that the JDBC[5] connection should connect to the ODBC connection called 'CSVMultiple'. The `driver` attribute specifies which specific driver to use when establishing a connection. In the example the jdbc to odbc reference implementation by Sun is used as is indicated by the driver name 'sun.jdbc.odbc.JdbcOdbcDriver'. The `username` and `password` attributes specify which credentials to use when establishing the connection. No username and password are provided in the example. The connection can also be given a `description` to describe the data source to connect to. This attribute is not used in setting up the connection to the database. It can however help users to identify their specific connection details when they receive a conversion definition. The description could for instance be 'Financial production database (Oracle)' or 'Local copy of SAP export'.

The root element from XES is the `Log` class. Because it is the root element in the XES format it is directly related to our `Mapping` class as the first event log element. The `Log` element in the event log can contain one or more classifiers for classifying events, as discussed in Section 2.2.2. Hence, our domain model has a `Classifier` class associated to the `Log` element. Each event classifier can be given a `name` and a list of event `keys` separated by spaces that are used as a classifier. In the example of Figure 4.2 the log element contains one classifier. This classifier is given the name 'Activity Classifier' and indicates that the keys 'concept:name' and 'lifecycle:transition' together form the event classifier.

The `Log` class also contains a `Trace` definition. Since each event log can only contain events related to a single process instance (see Section 3.2), we only need one trace conversion specification. The `trace` class contains one or more `Event` classes. Each `Event` class defines how to extract one event or a set of different events at once. It could for instance define how to retrieve the event information of creating an order. In Section 2.1.1 for instance the creation of an order would be one event conversion definition. Another event definition could be the extraction of certain records of the change log indicating an order line change. A single event definition could also define how to extract events from different activities at once. The data source might for instance include some sort of log where events are recorded. The event definition can extract certain events from

[5]JDBC is discussed in more detail in Section 5.1.

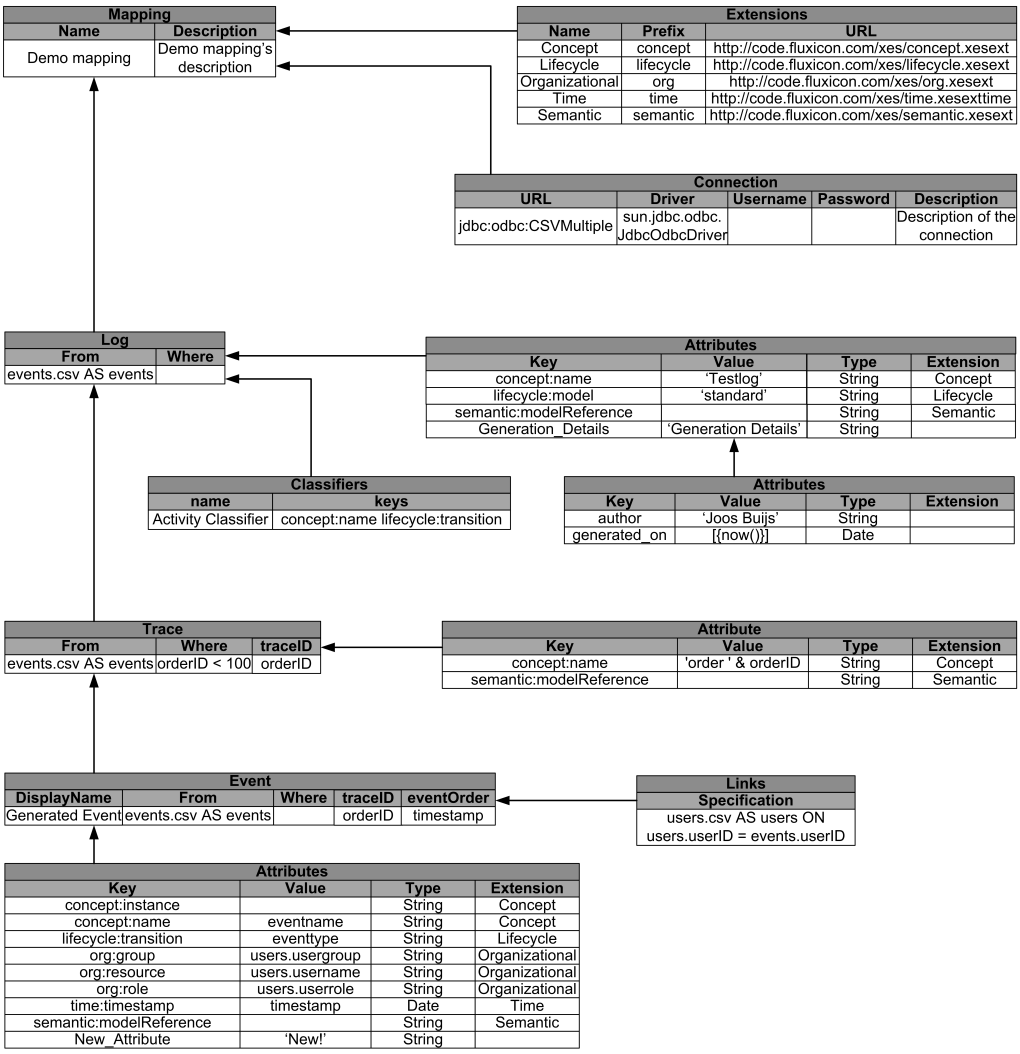


Figure 4.2: An instance of the class diagram for the example conversion definition.

this log to add to the resulting event log, possibly adding additional information. In the example of Figure 4.2 this is the case and only one event definition is necessary to extract all the events from the data source.

To be able to visually distinguish the different event definitions each event instance has a `displayName` attribute. The `displayName` can be used to identify different event definitions. These are used in the graphical user interface. In Figure 4.2 the event is displayed using the name 'Generated Event'. Other examples of display names are 'Create Event' or 'All start events'.

The `Event`, `Trace` and `Log` classes all belong to the `GeneralMappingItem` class, as can be seen in Figure 4.1. The purpose of the `GeneralMappingItem` class is to provide a generalization of these classes. This reduces the complexity of the domain model and allows the re-use of certain properties. The `GeneralMappingItem` class has two relationships to other classes. One to the `Attribute` class which defines that every `GeneralMappingItem` can have one or more attributes. In the example of Figure 4.2 there are several attributes defined for different items. Each `Attribute` instance contains a `key` which uniquely describes the attribute for the current `GeneralMappingItem`. The `Log` instance has four related attributes with keys such as 'concept:name' and 'Generation_Details'. The `value` attribute contains the actual conversion definition for the attribute. It defines what the value of this attribute is or how it can be retrieved

Table 4.2: Valid properties per mapping item.

	Log	**Trace**	**Event**	**Attribute**
From	X	X	X	-
Where	X	X	X	-
TraceID	-	X	X	-
EventOrder	-	-	X	-
Links	X	X	X	-

from the data source. This definition can for instance be a fixed value such as 'start' or 'Joos Buijs'. Another example is a value of 'table.column' to specify that the value of a certain column of a table provides the value of the attribute. In essence the value specification can be any valid specification in the SELECT-part of an SQL query. In the example of Figure 4.2 the log item has attributes with only fixed values such as 'Testlog' and 'standard'. The event item contains attributes with values from the columns of the data source such as 'eventname' and 'timestamp'. The trace item combines fixed and dynamic values using the "order' & orderID' value specification. More examples of value definitions will be shown in Chapter 6. The `type` attribute specifies the type of the value. These types are defined by the XES event log format. The attributes in Figure 4.2 are all of the type 'String' except for the 'time:timestamp' attribute of the event which is of type 'Date'.

Note that the `Attribute` class itself is also a specialization of the `GeneralMappingItem` class. Hence, instances of the `Attribute` class may contain attributes themselves. The 'Generation_Details' attribute of the log item in Figure 4.2 for instance has the sub-attributes 'author' and 'generated_on'.

The XES event log format works with extensions to allow a flexible specification of attributes that have a generally understood meaning. Extensions used in the conversion are stored in the `Extension` class. Each extension has a location on the internet where its specification is stored. This location is stored in the `URL` attribute. Once the `URL` attribute has been set, the values for the attributes `name` and `prefix` should be extracted from the extension specification. The name and prefix are stored to prevent repeating access to the extension definition. The extension definition also contains information on what attributes it defines and for which elements. Therefore, for each `Attribute` instance it is recorded to which `Extension` instance it belongs (if any). The reverse is also true: for each `Extension` instance all the `Attribute` instances defined by this extension are recorded. In the example of Figure 4.2 the mapping has 5 extension definitions, which are the standard extensions of XES. If an attribute is defined by an extension the name of this extension is shown in the 'Extension' column of the attribute.

Each `GeneralMappingItem` instance is also associated with the `Property` class. The `Property` class defines per `GeneralMappingItem` instance where the information is stored in the database. Which properties are used for different specializations of the mapping items is shown in Table 4.2. In the example of Figure 4.2 these properties are shown within the `Log`, `Trace` or `Event` items.

The `from` property describes what the start table is for this mapping item, e.g. 'table1'. This corresponds to the FROM-part of an SQL query. In Figure 4.2 the `from` property is 'events.csv AS events' for all items. This indicates that the table 'events.csv' will be used and that we will use the alias 'events' to refer to this table. Multiple tables can be joined together on two different ways. More experienced users can specify the join in the `from` attribute. An alternative is to define one or more instances of the `Link` class. This link class specifies how a link is made to a certain table from the start table. An example of a link specification is 'table2 ON table1.field1 = table2.field2'. This specification joins 'table2' to 'table1' (which is defined in the `from` attribute) on the value of 'field2' respectively 'field1'. For most users and situations the `Link` class provides enough expressive power to join tables. The benefit of using the `Link` class is that link specifications define how to connect a table to the base table. This is easier than the definition of a full `from`

specification combining multiple tables at once. The event instance in Figure 4.2 has such a link associated with it. The link specifies that the table 'users.csv' is connected to the table 'events' on the field 'userID' of both tables.

The next property is the `where` attribute which can be used to select which records are used. This can be used for instance to limit the number of traces or events that are included in the event log. An example specification is 'table1.field1 BETWEEN 0 AND 1000'. For the log item this property can be used to select a particular record for information. In the example of Figure 4.2 the `where` property is used for traces. Only those traces that have an orderID below 100 will be included in the event log.

Events need to be related to a trace instance. To achieve this, each trace specifies a `traceID` that is unique for each trace. This can for instance be the order identifier if the trace corresponds to an order, as is the case in Figure 4.2. To relate events to traces, events also specify a `traceID` value. This value specifies to which trace instance the event is related. So if the trace corresponds to an order, the `traceID` of an event should result in the order identifier of the order to which the event is related. In Figure 4.2 the event also defines the 'orderID' to be the trace identifier. The `traceID` is also used to select only those events that relate to traces that were extracted.

Furthermore, events are ordered. To make sure the events in the event log are ordered correctly the value to order on can be specified in the `eventOrder` attribute. In many cases the order value will be something like an identifier or a timestamp, e.g. 'table1.ID' or 'table1.time'. By default events are ordered ascending but by adding the keyword 'DESC' events will occur in descending order in the event log. In the example of Figure 4.2 events are ordered on the timestamp value.

Please note the following:

1. Attribute values for the log item can only be taken from the first record in the query result. If specific data from the data source should be used, care should be taken to select the correct record;
2. For traces and events each resulting record is considered a new instance of a trace or event respectively;
3. By default only events are included for which the related trace is already extracted.

4.3 Conversion Visualization

To improve usability further, the conversion definition can also be visualized. This visualization will show the general mapping items log, trace and events with their attributes. It will also display the tables and columns used in the conversion definition. The connection between the attributes and the columns they use to retrieve their value are also displayed by arrows. This visualization enables the user to get an overview of the conversion. It can also serve as a starting point for a discussion with domain experts.

A first impression of the visualization is shown in Figure 4.3. The visualization shows the conversion definition of Figure 4.2 in a more readable way. The general mapping items are displayed as circles in a tree layout with the log element as root. This corresponds to the tree structure used in the application. In the visualization all attributes are shown and not just those with children. On the right of the general mapping items are the tables and columns used. Not all tables in the data source will be shown. The data source might contain many tables of which only a few might be used. Displaying all tables of the data source would unnecessarily clutter the visualization. For the same reason not all columns will be shown. Only those columns that are actually used will be included.

In the example of Figure 4.3 it is clear to see that the `time:timestamp` attribute of the event definition gets its value from the `timestamp` column of the `events` table. It is also clear to see that the three attributes of the organizational extension all use data from the `users` table. None of the log attributes use data from the data source. Of the trace only the `concept:name` attribute uses data from the data source.

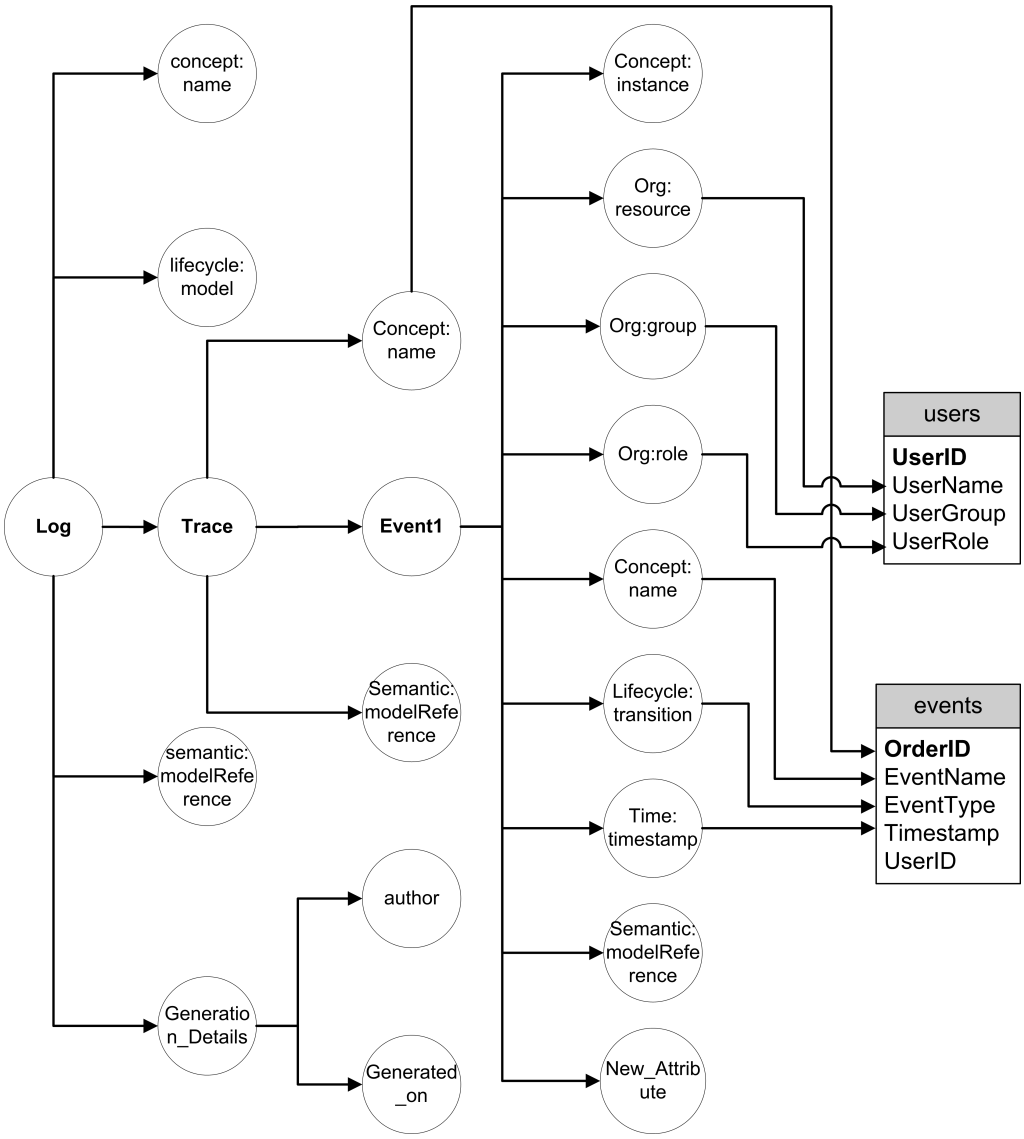


Figure 4.3: Visualization sketch.

4.4 Cache database

The conversion can now be defined and visualized, the next step is to allow the execution of the conversion. To execute the conversion several steps need to be performed. The three steps used in our approach are shown in Figure 4.4. 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 the XES event log. Each of these steps is explained in more detail in Chapter 5. In this section we will discuss the reason and structure of the cache database in which the intermediate results are stored.

There are multiple reasons to use an intermediate cache database. The main reason is that it is more efficient to run a few queries with many results than executing a lot of queries each returning only one or a few records. Especially when many joins on large tables are included, the difference is significant. Note that the number of events that need to be retrieved from the data source can be large. Running thousands of similar queries will take more time than running a

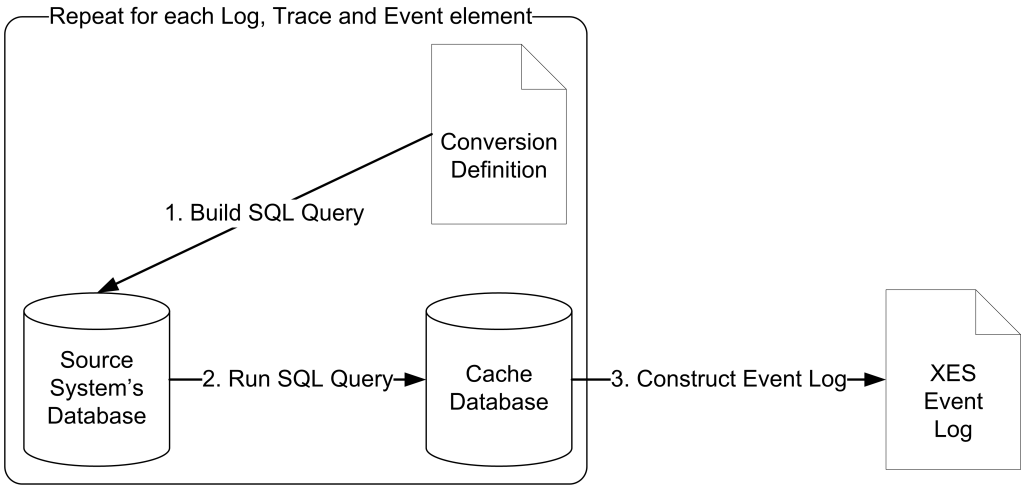


Figure 4.4: The three execution phases of the application.

more general query resulting in thousands of records.

Another reason to use an intermediate database is that the XES event log is most efficiently written in the correct sequence at once. It is very inefficient if first all the traces are added to the event log and later on the events for each trace need to be inserted in between. Therefore, it is best to first collect and store all the information required and then write it to the event log in a sequential stream.

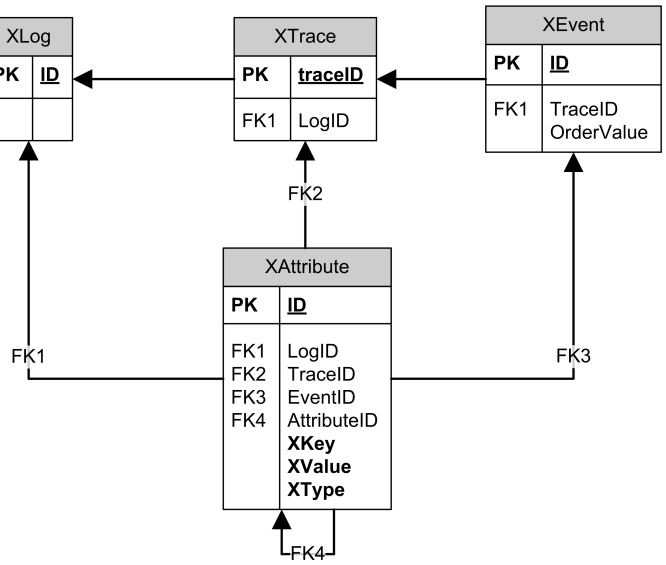


Figure 4.5: Intermediate database structure.

The structure of the intermediate database is shown in Figure 4.5. Each square represents a table within the database. The 'PK' keyword indicates a primary key, a field that uniquely identifies a record. The 'FK' keyword specified a foreign key. Foreign keys specify that the value of that column relates to a primary key in another table. This is also indicated by the arrows between tables. This database structure enables the storage of a complete XES event log except for the global attributes and the event classifiers. The global attributes are calculated just before step 3 is executed and are then added to the event log. The event classifiers are added directly from the conversion definition.

The `XLog` table contains all the log instances. Since each event log only contains one log element there will be only one. The reason for including a separate table for a single log entry is that there are attributes in the `XAttribute` table related to the log. Both the `XLog` and the `XAttribute` tables contain an automatically created numeric identifier in the `ID` field. Per attribute in the `XAttribute` table it is stored to which log, trace, event or other attribute it refers. These references are stored in the `xxxID` fields of the `XAttribute` table. Each field refers to the type of item the attribute relates to. Please note that only one of the four fields should contain a reference to another record. Unfortunately, this structure can not be avoided due to the structure of the XES elements. Especially the fact that attributes can relate to different items but can also contain other attributes themselves is hard to express elegantly in a relational database structure. Besides a reference to its parent the `XAttribute` table also contains information on the attribute itself. Information about the key of the attribute is stored in the `XKey` field. The value is stored in the `XValue` field and the type in the `XType` field.

Traces are stored in the `XTrace` table. The reference to the log item is stored in the `LogID` field. The trace identifier is stored in the `traceID` field and contains the value that was the result of the `traceID` definition for the trace in the conversion. The `XEvent` table stores events with an automatically generated identifier in the `ID` field. The reference to the trace is stored in the `traceID` field. To be able to order the events per trace the value to order on is stored in the `orderValue` field.

Figure 4.6 shows an instance of the cache database of our running example used in this chapter. This snapshot shows the cache database just after steps 1 and 2 are performed for each element. Using this data the XES event log will be constructed in the third and final step of the conversion execution. The figure shows four tables of which the `XAttribute` table is the largest. Every attribute of each of the items is stored in this table. The first three attributes in the `XAttribute` table are related to a record in the `XLog` table. This is indicated by the value in the `LogID` column of the `XAttribute` table. Each conversion will only contain one log record as specified in the XES format. The three attributes that relate to the log record are the attributes that were defined for the log item in the conversion definition.

If we recall the conversion definition of Figure 4.2, the 'Generation_Details' attribute of the log item contained two sub-attributes. The next two attributes, with ID's 3 and 4, therefore refer to the attribute with ID 2, which is the 'Generation_Details' attribute. The next three records in the `XAttribute` table, with ID's 5, 6 and 7, each refer to a record in the `XTrace` table. Each attribute contains the value for the 'concept:name' attribute of one of the traces. Each trace in the `XTrace` table is related to log 0. Attributes 8 until 14 are all related to the first event in the `XEvent` table. The next 7 records in the `XAttribute` table are related to the second event and so on for all 17 event entries. Each event entry in the `XEvent` table contains a reference to the trace they belong to in the `traceID` column. The `orderValue` column stores the value on which the events are ordered during event log creation.

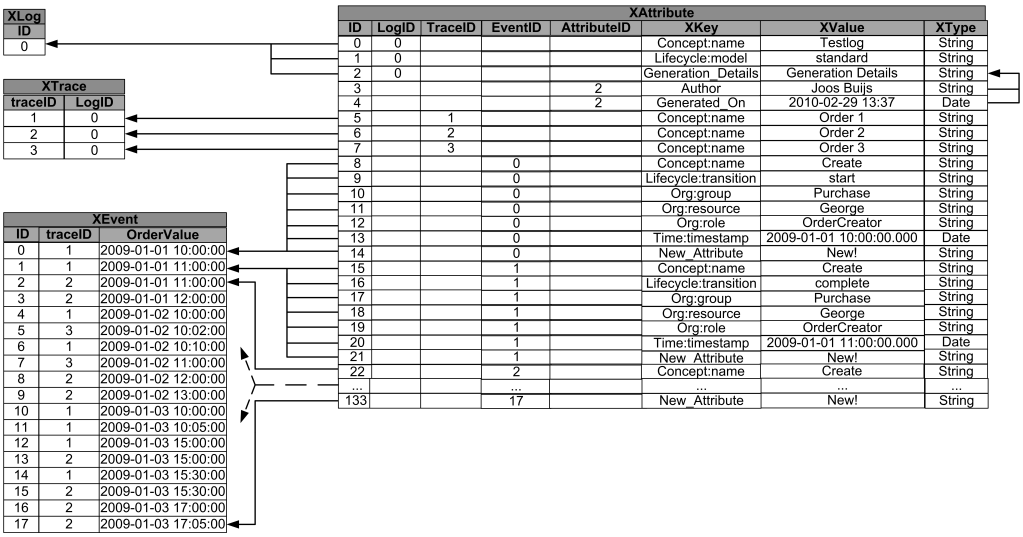


Figure 4.6: Intermediate database instance.

4.5 Conclusion

In this chapter the foundation for the implementation of the application has been laid. The key requirements of the application have been investigated. A domain model to store the conversion definition was also defined. This domain model stores all the information required to be able to execute the conversion. To aid the user in understanding and investigating the conversion definition, a visualization approach has been defined. This visualization shows how the different elements of the event log are related to the tables and columns of the data source. To be able to execute the conversion an intermediate database is required. This database is used to temporarily store the event log data before the event log is constructed. Now that all the main aspects of the application have been investigated, the actual implementation of the application can begin. The implementation details are discussed in Chapter 5.

Chapter 5

Solution Implementation

This chapter discusses the implementation of the application of which the functionality was described in Chapter 4. The application has been given the name 'XES Mapper', abbreviated to 'XESMa'. Section 5.1 discusses the most important decisions made regarding the implementation. The resulting internal structure of the application is discussed in Section 5.2. Section 5.3 describes the graphical user interface of the application. The execution of the conversion is discussed in Section 5.4.

5.1 Main Implementation Decisions

This section discusses some general implementation decisions made. One of the first decisions to make was what programming language to use. It was rather clear that this would be Java. Both the ProM and ProM Import frameworks are written in Java. Furthermore, the only library written to interact with XES event logs, the OpenXES library, is also written in Java. Last but not least, Java can be run on almost all operating systems.

Initially the idea was to create the prototype as a plug-in for Eclipse[1]. Eclipse is a software development environment including a plug-in system to extend it. The benefit of writing a plug-in for Eclipse is that basic things such as loading and saving and basic graphical user interface elements are provided. Furthermore, by using the Eclipse Modeling Framework[2] (EMF) development should be quick and easy. Unfortunately, the Eclipse framework for writing plug-ins is complex and not very well documented. It also assumes a certain structure of the application and the graphical user interface. Furthermore, Eclipse can become slow and unstable when many plug-ins are installed. After 2 weeks of trying to adapt the plug-in generated by EMF to our wishes, with the help of an expert, it was decided to abandon this idea. In the 2 weeks that followed, the graphical user interface was build from scratch using basic Java objects. Starting from scratch also has benefits. For instance the application code is simpler and better structured and hence easier to understand. Furthermore, it provided more freedom in defining the graphical user interface. There is also no overhead from the Eclipse framework which increases performance. In the end this solution turned out better than an Eclipse plug-in would.

The application is developed following the model-view-controller architectural pattern [7, p. 239]. This pattern isolates the domain logic, the graphical user interface and the data aspects of the application. This is visualized in Figure 5.1. The model contains the mapping definition. This is stored in the domain model as described in Section 4.2. The domain logic is provided by the controller classes. The controller classes handle most of the requests made by the user via graphical user interface. The graphical user interface provides interaction with the model and controller to the user. The layout of this interface is discussed in Section 5.3. The benefit of using this pattern for the separation of functionality is that it makes the application more flexible. For

[1] See http://www.eclipse.org
[2] See http://www.eclipse.org/emf

instance the application can also be used without the graphical user interface. This can be done by directly accessing the controller and model classes via Java code. The graphical user interface can also easily be replaced without affecting the controller and model classes. The same holds for the model classes. At the moment the mapping definition is stored in an XML file. By changing only the model classes the storage format can be changed. Section 5.2 discusses this separation and the functionality of each section in more detail.

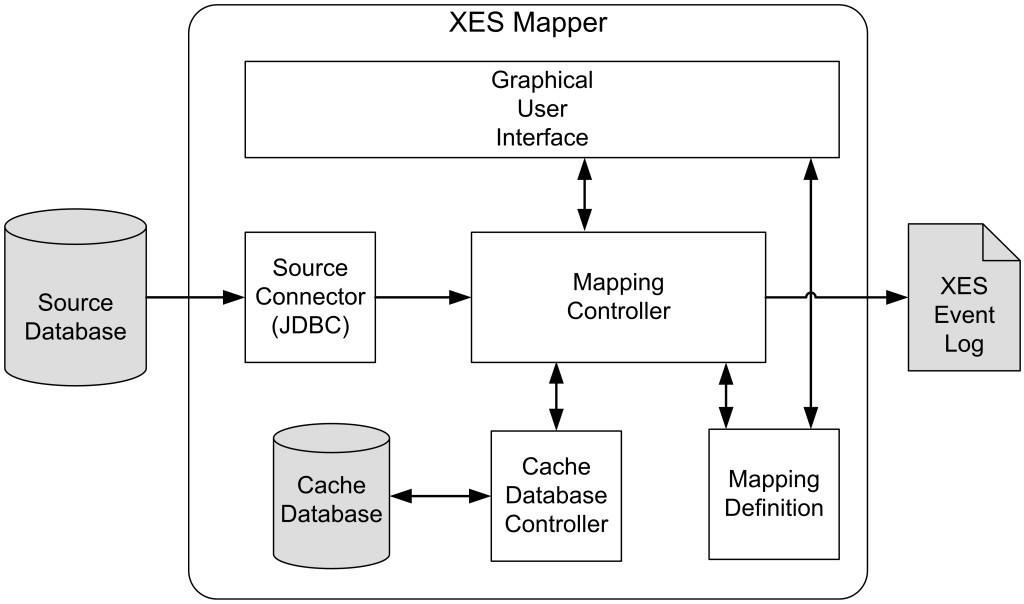


Figure 5.1: The environment of the application

To connect to the source database we use the JDBC[3] application programmers interface (API). We have chosen this interface because this API enables access to a variety of databases of different database vendors. JDBC is the main supported method for establishing connections to databases in Java. It is also possible to connect to ODBC data sources. Furthermore, the interaction through JDBC is standardized which means that programming is simplified. JDBC also provides access to non-database input formats such as comma-separated-values files and XML files. This can be done using JDBC or ODBC drivers for these file types. Hence, by using JDBC we allow access to a wide range of data sources. At the same time the standardized interface allows for efficient and generalized code development.

The conversion execution makes use of an intermediate database to store the data as discussed in Section 4.4. As discussed there, there are several reasons for the use of an intermediate database. The main reason is to limit the number of queries run on the data source. Furthermore, the event log should be written in sequence. Since not all intermediate results will fit in the system's memory we store it in an intermediate database. For the implementation of this database Java DB[4] is used. Java DB is Sun's supported distribution of Apache's Derby database. This database implementation uses little resources and is mainly developed for internal use by applications. There are other alternative such as the SQLite, HSQLDB and H2 database systems. Since Java DB is supported by Sun, the main developer of Java, we have chosen for this implementation. XESMa can however be adjusted to use a database from another vendor if required.

[3]See `http://java.sun.com/products/jdbc/overview.html`
[4]See `http://developers.sun.com/javadb`

5.2 Internal Structure

This section discusses the internal structure of the application on the level of Java packages and classes. As mentioned in Section 5.1 the application is structured following the model-view-controller pattern. This is clearly visible in Figure 5.2 by the division in three packages. Each of the larger three rectangles represents a package. Packages of an application are grouped by a common prefix. All packages of this application have a prefix of 'org.processmining.mapper'. Each package contains a group of one or more classes or sub-packages. Each class also contains variables and functions, but these are not shown.

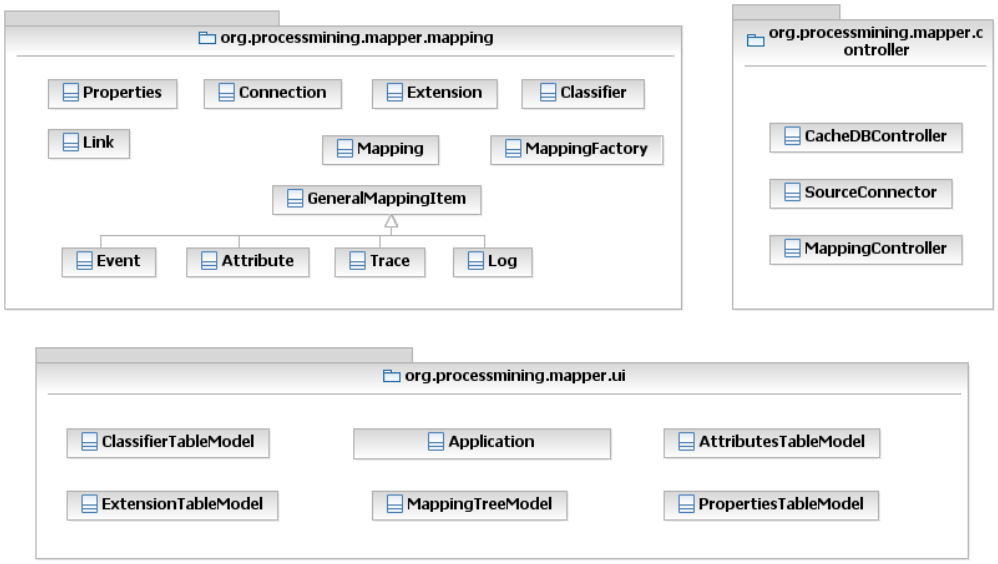


Figure 5.2: Class diagram of the application.

The top left rectangle contains the model section. The package name is 'org.processmining.-mapper.mapping'. The classes inside the package almost all occur also in the domain model as shown in Figure 4.1. For a detailed discussion we refer to Section 4.2. One addition is the 'MappingFactory' class. This class provides functions to create new instances of the model classes. The factory sets some default settings in these new instances for convenience reasons.

The 'org.processmining.mapper.ui' package provides the user interface (hence the abbreviation 'ui'). The main class of this package is the 'Application' class. This class builds up the whole graphical user interface. It also defines the actions of buttons and of other events. It also interacts with the model and controller classes. For some model classes certain intermediate translating classes needed to be specified to display the contents. For example for the mapping tree on the top left in Figure 5.3. This tree view on the model data is constructed by the 'MappingTreeModel' class. It takes the domain model contents and provides access to certain elements of it in a standardized tree way. This allows the use of a standard Java tree element to display the tree structure. The four other classes in this package provide interfaces to display the classifiers, extensions, attributes and properties as tables in the user interface. Each model class translates the tree or table interface to the internal data model used. For instance, it defines how many columns a table has, what the headings are and which fields are editable.

The user interface also interacts with the controller classes. These classes contain most of the business logic of the application. The controller package on the top right contains three classes. The 'SourceConnector' class provides access to the data source. In our application this is currently limited by using the JDBC libraries. In the future this class could be extended or specialized to handle more types of data sources. One of the other classes is the 'CacheDBController' class. This class handles everything related to the intermediate database. This class initializes the

database and provides easy ways of inserting and extracting information. By separating this logic the intermediate database implementation can be changed without affecting the rest of the application. The main controller class is the 'MappingController'. This class handles the saving and loading of conversion definitions, interpretation of XES extensions and the correct handling of all kinds of messages. The most important function of this class however is executing the conversion. This is discussed in more detail in Section 5.4.

5.3 Graphical User Interface

Figure 5.3 shows the graphical user interface of the XES Mapper application. The interface is divided in three sections. The bottom half of the window is reserved for general settings. These include the project name and description, connection settings, what XES extensions to include, settings for the execution of the conversion, a message console and a visualization of the conversion definition. In the figure the project settings tab is shown with the name and a brief description of the loaded conversion. The top half of the window is used for the conversion definition itself. The top left half is used to browse the conversion definition. It displays the main event log items such as the log, trace and events items. These items are displayed in a tree structure. In the figure there is one event definition with the display name 'Generated Event'. The tree also contains a log attribute named 'Generation_Details' which has child-attributes. The top right half is used to define the attributes and properties of the item selected on the left. In the figure the attributes of the selected event are shown. Each part of the graphical user interface will be discussed in more detail in this section. During this discussion the simple example discussed in Section 4.2 will be used. The definition will result in the domain model instance as shown in Figure 4.2. More complex conversions will be defined in Chapter 6. The conversion discussed in this section is rather straightforward and is used to demonstrate the main functionality of the application.

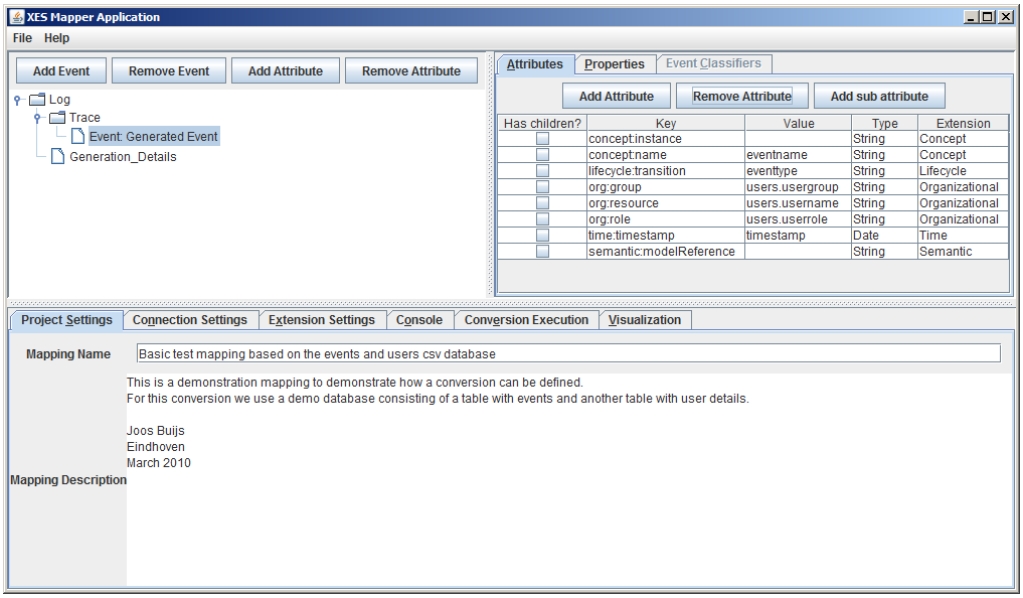


Figure 5.3: Graphical user interface of the application.

5.3.1 General Settings

As mentioned before, the bottom half of the window is used for the general conversion settings. The first tab is the `Project Settings` tab as shown in Figure 5.3. This tab contains two fields, one for the name of the conversion and one for a more detailed description.

The next tab is shown in Figure 5.4 and is the `Connection Settings` tab which is used to define the connection to the source database. The main setting for the connection is the `URL` which locates the database and may provide additional properties for the connection. The `Driver` field is used to tell the application what JDBC driver to use when trying to connect to the database. For some connections credentials need to be provided by entering a valid `username` and `password`. Additionally a `description` of the connection can be provided. This description can be used to describe the target data source. This is especially useful when the connection to the data source fails. When the user did not define the conversion himself, the description should help him identify the data source. This should allow the user to correct the connection details for their situation. The connection can be tested using the 'Test Connection' button. If the connection fails an error message is displayed and the user should correct the settings.

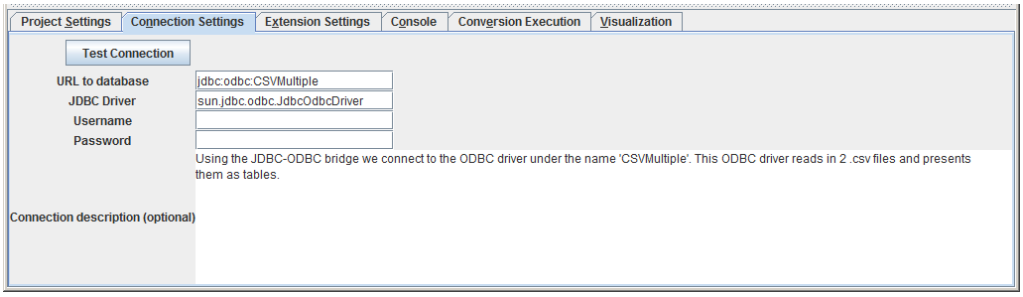


Figure 5.4: Connection settings tab.

The third tab is the `Extension Settings` tab as shown in Figure 5.5. The purpose of this tab is to handle what XES extensions are to be included in the event log. By including an extension the attributes defined by the extension are also added to the conversion definition. Figure 5.5 also shows the popup window that appears when the `Add Extension` button is pressed. In this popup the location of the extension definition can be defined. In most cases this is an URL but it can also be a local file. When the `OK` button is pressed the application loads the extension definition. It then updates the domain model by adding all attributes defined by the extension to the log, trace and event instances. The extension is also added to the list of included extensions. In the figure all 5 standard extensions are loaded. The `URI` column displays the location of the extension. The `Prefix` column displays the prefix used by this extension in the attribute keys. The logical name of the extension is shown in the `Name` column. Please note that the values in the table can not be edited. Extensions can only be added or deleted. Changing the URL, prefix or name of an extension therefore is not possible. When an extension is deleted all attributes defined by the extension are also removed.

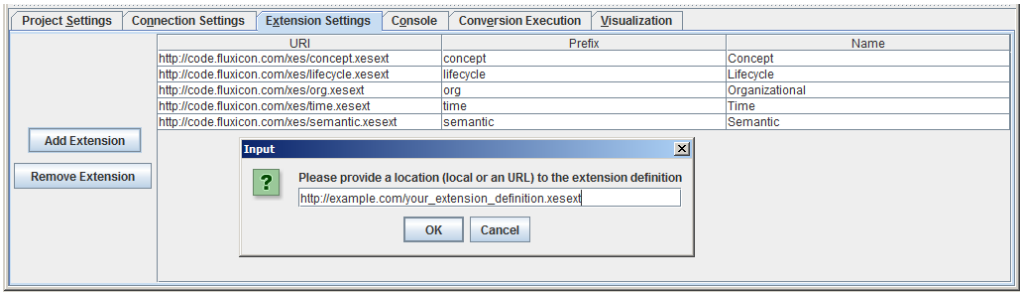


Figure 5.5: Extension settings tab.

Figure 5.6 shows the console after a conversion execution. During execution different types of messages regarding the execution are outputted to the console. There are 5 levels of messages. Listed from more to less important these are: error, warning, progress, notice and debug. In the

example all messages but debug messages are shown since the minimal level is set to 'notice'. It is advised to only set the message level to 'debug' during testing. The performance will suffer dramatically if debug messages are outputted during large executions. Each message is preceded with a date and time of occurrence. The message type is also added before the message. The type of message can also be seen from the text color of the message.

The conversion execution is divided in several steps, as can be seen in Figure 5.6. To indicate the progress of the conversion, a message is displayed on the console when a next step commences. The steps the application makes are the initialization of the application, extracting the information for the log, trace and events from the data source and writing the XES file. More details on each of these steps are provided in Section 5.4. It is impossible to give a time prediction for each step because of the many factors that influence the execution time. Examples of influencing factors are the type and structure of the data source, the conversion complexity and the computer configuration. Progress is also indicated by the progress bar at the bottom right of the figure. This bar indicates what fraction of the total conversion has been executed.

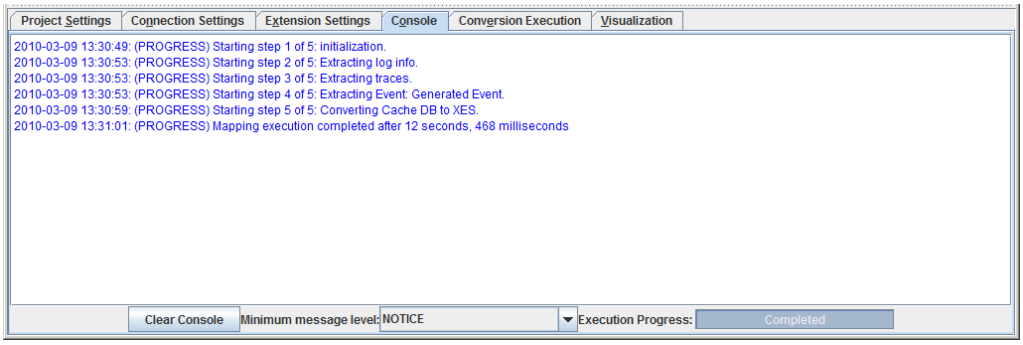


Figure 5.6: Console tab.

The last tab is shown in Figure 5.7 and allows for executing the conversion and the specification of execution options. The execution can be started with the `Execute Conversion` button. The first option is the number of traces to include in the event log. This can be used to limit the size of the resulting event log. It can also be used to perform a test run for verification purposes. One can for instance start with a small number of traces. This will shorten the execution time and reduce the size of the generated event log. Furthermore, debug messages can be enabled in the console for a more detailed investigation. This allows for a detailed investigation of the conversion execution and the resulting event log. If the resulting event log is correct after examination, more traces can be included. The event log can also be written in the MXML format instead of XES. This can be done by checking the first checkbox. The event log can also be compressed in a zip file directly by checking the 'generate zip' checkbox. The location of the intermediate database can also be chosen. Since this database can become very large a location where enough space is available should be selected. If the location needs to be changed, the button `Change CacheDB Location` can be pressed. The current location is shown in the text field below the button. The location and name of the resulting event log can also be chosen. If the name and/or the location of the log should be changed the `Change event log location` can be pressed. The current file name and location is shown below the button. Please note that an existing event log is overwritten without warning!

The final tab, that of the visualization, is discussed in more detail in Section 5.3.3.

5.3.2 Conversion Definition

The conversion definition is specified in the top half of the window as can be seen in Figure 5.3. The left part allows for navigation through the various elements. Figure 5.8 show this part in more detail. The root element is always the log element followed by the trace element. Below the trace element one or more event elements can be added. Adding and removing events is done

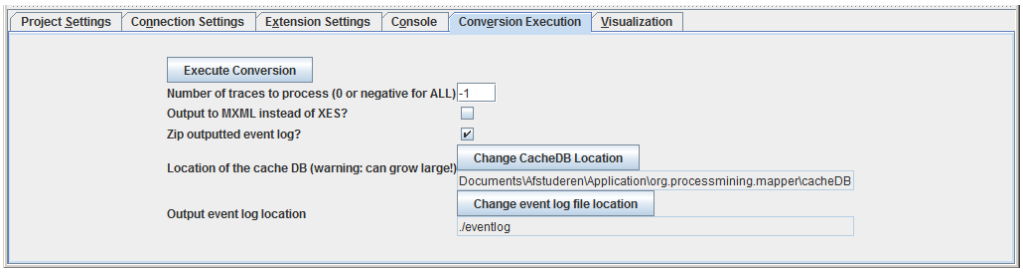


Figure 5.7: Execution settings tab.

with the corresponding buttons. In this tree view attributes that contain child-attributes are also shown. These will be discussed later on in this section.

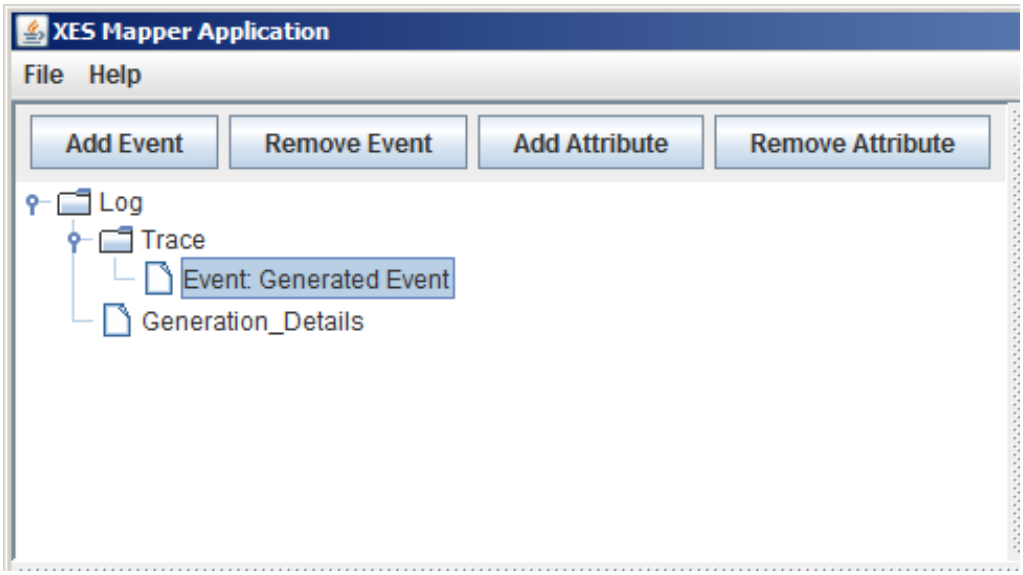


Figure 5.8: Top left part of the application: log elements in a tree structure.

Each element in the event log can have attributes. Some of these attributes can be defined by the extensions discussed earlier. Figure 5.9 shows the attributes as defined for the event selected in Figure 5.8. As can be seen there are several attributes defined. Most of them are defined by an extension as shown in the 'Extension' column. For attributes defined by an extension only the 'value' can be defined. The last attribute, called 'New_Attribute', is added manually. This can be done by pressing the `Add Attribute` button. For manually added attributes the key and type can also be defined. This is not possible for attributes defined by an extension. As can be seen 5 different data types are available. The value for our manually added attribute is 'New!' which is a fixed value. This means that each event in our event log will have an attribute called 'New_Attribute' with a value 'New!'. Attribute values can also be filled in from the data source. In this example the value for the 'time:timestamp' attribute is retrieved from the 'timestamp' field in the current table in the database. The current table can be defined in the `Properties` tab which will be discussed later.

For attributes of the type 'Date' a special function is available. Using the notation '[{format}]' the time format of the input can be specified. This format is used when the input is converted to a date and written in a standardized format in the event log. Since date formats can take many different forms, no generic algorithm can be made to interpret a date in any format correctly. The format provided should follow the Java date format as described at `http://java.sun.com/javase/6/docs/api/java/text/SimpleDateFormat.html`. The timestamp of '2-1-2009 10:00' for

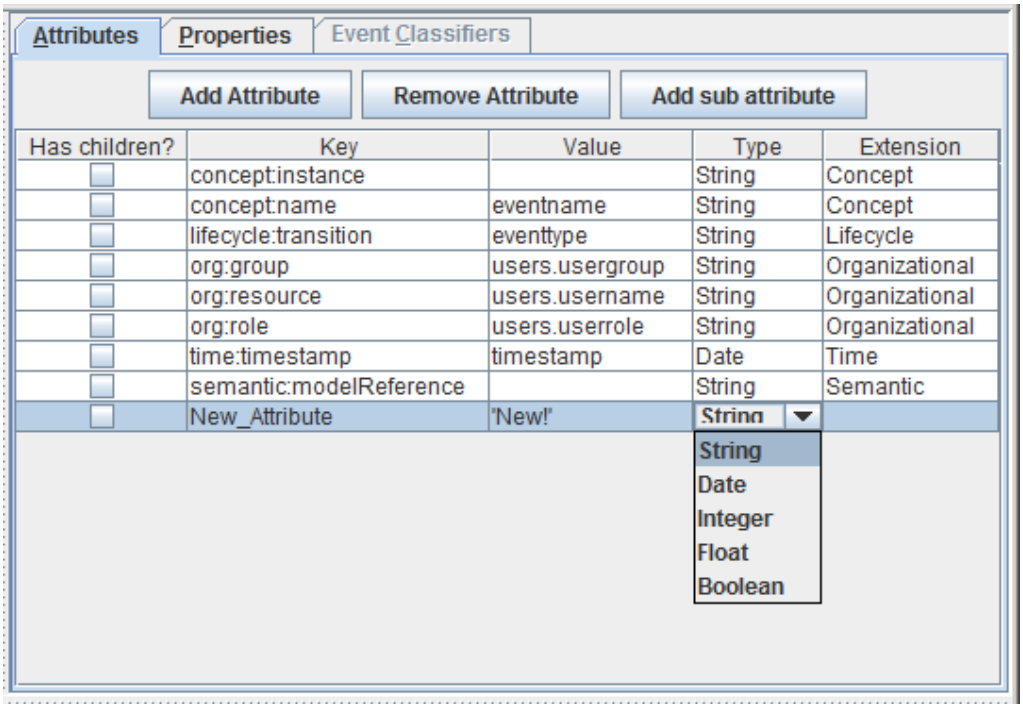


Figure 5.9: Attributes of the event definition.

instance is formatted using the format notation 'dd-MM-yyyy hh:mm'.

The value of an attribute can also consist of more complex definitions. Anything that is supported in SQL by the JDBC driver used is valid for the value specification. Unfortunately, during testing it was found that in certain cases functions in the query were not recognized as such, see Section 6.3.1. Therefore two functions were re-implemented by our application. The first is the current date and time, which might be used in the log description. This can be achieved by entering '[{now()}]' in one of the attributes. Another function that is re-implemented is the substring function. By entering '[{substr(pos,len)}]' a substring of the result is put in the log. The first parameter *pos* is the position of the first character of the string to include, starting at 1. The *len* parameter is optional and indicates the length of the substring. If *len* is not provided the string starting at *pos* is returned.

Attributes can also have 'child-attributes' which provide more detail on the current attribute. If an attribute has child-attributes the checkbox in the column 'has children?' in the attribute table is checked. Furthermore, the attribute also appears in the tree view as shown in Figure 5.8. In this figure the 'Log' element has an attribute 'Generation_Details'. This attribute can be selected in the tree and its child-attributes will be shown in the attribute list. These child-attributes are treated just as normal attributes. They can even be given child-attributes themselves.

Log, trace and event elements also have properties which are used during the conversion. These properties were discussed in Section 4.2. Figure 5.10 shows the properties of the event selected in Figure 5.8. The `from` property specifies the base table in the database. This can be compared with the 'FROM' part of an SQL query. In the example the base table is 'events.csv'. For convenience we renamed this table to 'events'. Other tables used in the conversion can be joined together using links. In this example there is a relation from the 'events.csv' table to the 'users.csv' table. This relation is made by connecting the 'userID' field from the 'events.csv' table to the 'userID' identifying field of the 'users.csv' table. This relationship can be defined in the 'link' property as is shown in Figure 5.10.

Each event should be related to a specific trace for which it occurred. Each trace has a unique identifier as defined in the 'traceID' property of the trace item. In this example trace instances

are extracted from the 'events.csv' table without additional attributes. Each unique value for the 'orderID' column is considered to specify a new trace instance. For events the 'traceID' property defines the identifier of the trace it belongs to. In this example the trace identifier for events is conveniently defined by the 'orderID' column in the 'events.csv' table.

The 'where' property allows for defining a selection of records to include as traces or events. For traces this field can be used to only include certain traces. Only events related to the extracted traces will be included in the event log. In the example all records are considered to be events and no clause has been added. A selection could be made for instance to only include events that occurred within a certain time frame. More complex selection criteria for events are used in one of our case studies as will be discussed in Section 6.1.

Furthermore, events can be ordered on a certain value. In this example events should be ordered chronologically through time. Therefore the 'timestamp' attribute of the 'events.csv' table is used for ordering.

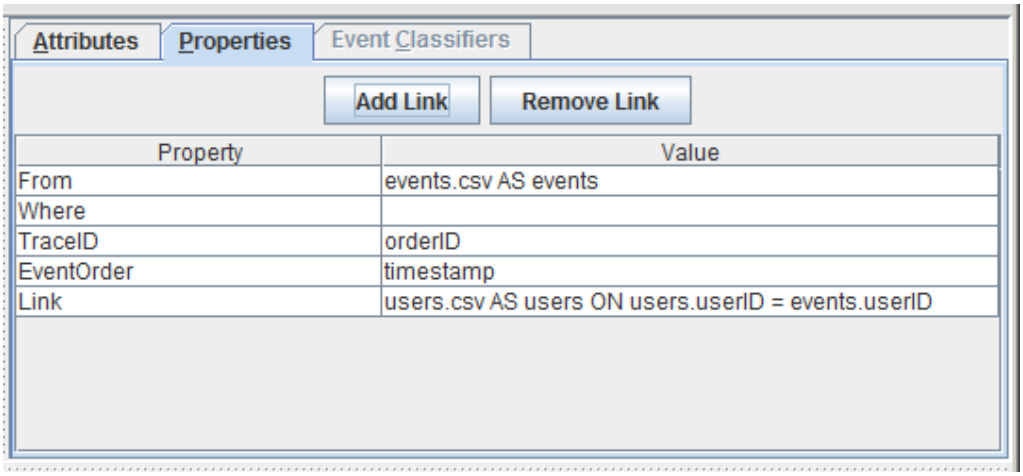


Figure 5.10: Properties of the event definition.

There is one item left that should be defined. As discussed in Section 2.2.2 the XES event log format defines event classifiers. These classifiers are used to identify events and make them comparable. Each event log should contain its own classifier definitions. In Figure 5.11 the classifier defined on the log item is shown. This is actually the default classifier which is added automatically by the application since it is the most common one. This classifier defines that each event is uniquely identified by the combination of the event name and the lifecycle transition attributes. Classifiers can be edited or removed and new classifiers can be added.

The menu items of the application were not discussed so far. The file menu provides functionality such as saving the conversion definition. Naturally, saved conversions can also be loaded into the application. The file menu also provides an exit function to close the application. The help menu provides access to information about the version and author of the application via the about option.

5.3.3 Conversion Visualization

As discussed in Section 4.3 the conversion definition can also be visualized. The visualization tab is shown in 5.12 with an example visualization. General mapping items are represented by elliptic shapes in a tree layout. The tables used in this example are shown to the right of the tree.

The user has several options for the visualization. The first is the 'Clear and Rebuild Visualization' option. This option clears the entire visualization and rebuilds it, discarding all layout information. This option should be used when the conversion definition changed drastically. The 'Update Visualization' is the second option and will add new conversion definitions to the model.

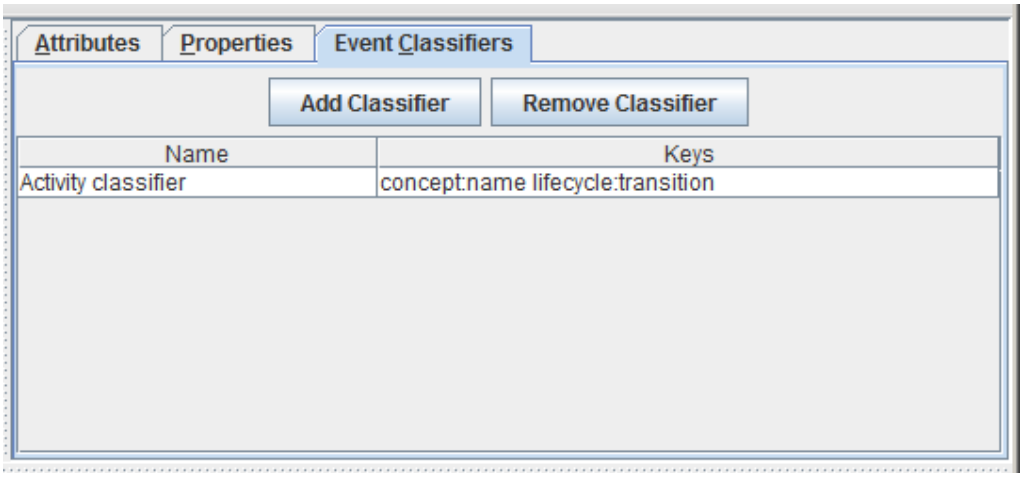


Figure 5.11: Event classifiers as defined at the log element.

The advantage of this option is that no connection to the data source is required. A disadvantage is that only additions are processed, removed attributes will remain in the visualization. The third option is to export the visualization as an image. Supported formats are scaled vector graphics (svg), portable network graphics (png), JPEG (jpg) and bitmap (bmp).

For the implementation of the visualization the JGraph[5] Java library (version 5.13.0.3) is used. This library allows for easy creation and configuration of graphs. The general mapping item class for instance extends a specialized class to display the instance in an elliptic shape. The table class of the domain model extends another specialized class. This specialized class displays all the columns of the table and attaches the lines from a specific column to an attribute.

The visualization is build in several stages. In the first stage the general mapping items are initialized for visualization. This means that each item is added to the visualization in an elliptic shape. Furthermore, attributes are connected to their parents, as are the events to the trace item and the trace item to the log item. In the second stage for each attribute the table(s) and column(s) used in the value specification are detected. This is done in two ways. The first is to build and run the query for the mapping item. The JDBC library provides meta data about the columns and tables used for each attribute. Unfortunately, this does not work if an attribute consists of multiple columns or of a combination of fixed and column values (e.g. 'order is `orderID`'). Therefore, if the first method fails, a second, more error prone, method is used. All the column names of the tables used in the query are retrieved. Then it is checked whether a column name appears in the value specification of the attribute. If this is the case then it is assumed that this column is actually used in the value of the attribute. Once all the attributes have been examined in this way this is processed in the visualization. Each relationship is visualized by adding a line from the attribute to the column in the table that it uses.

[5] see `http://www.jgraph.com`

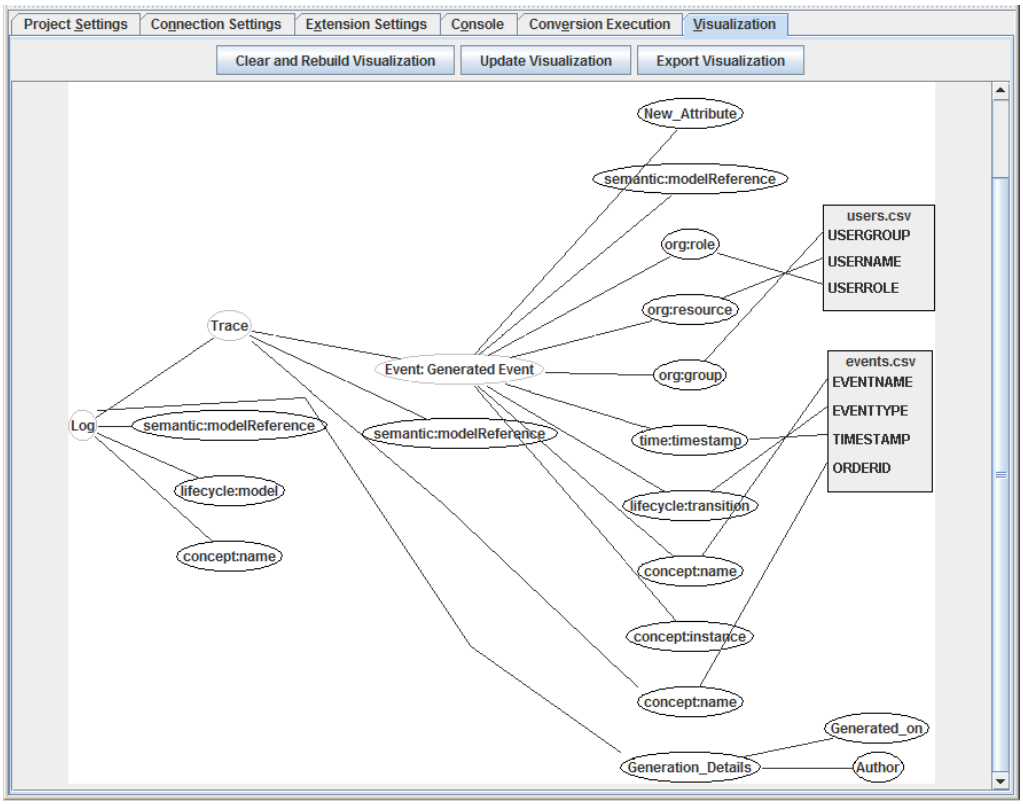


Figure 5.12: Conversion visualization.

5.4 Conversion execution

The conversion definition is executed in three steps as shown in Figure 5.13. The first step is to create an SQL query for each log, trace and event instance from their conversion definition. The total number of queries is therefore equal to the total number of event definitions plus 2 (one log and one trace definition). The second step is to run each of these queries on the source system's database. The results of these queries need to be stored in an intermediate database. The intermediate database was discussed in Section 4.4. The third step is to convert this intermediate database to the XES event log. This section will discuss each of the three steps in more detail.

5.4.1 Building the Query

As said, the first step is to convert the conversion definition to SQL queries. Each of these queries extract the attribute values for one of the mapping items from the data source. This query uses the properties as defined with the item to correctly build this query. Listing 5.1 shows the resulting SQL query for the event conversion definition as discussed in Section 5.3.2. The main purpose of the 'SELECT' part of the query is to extract the values for the different attributes. The first two definitions '`orderID AS [traceID]`' and '`timestamp AS [orderAttribute]`' are special in this query however. The first definition gets the value of the trace identifier the event belongs to. In this case the 'orderID' column is selected as the 'traceID' value. As mentioned before it is required to connect an event to a trace. The second definition is the 'orderAttribute' or the value to order the events on. In this case the timestamp is used. Both these values are stored in the intermediate database and used during the conversion to the event log. For the log element no special columns are added. The trace element only has an additional 'traceID' column besides the attribute columns. The rest of the parts are simply formed by the attribute specifications as

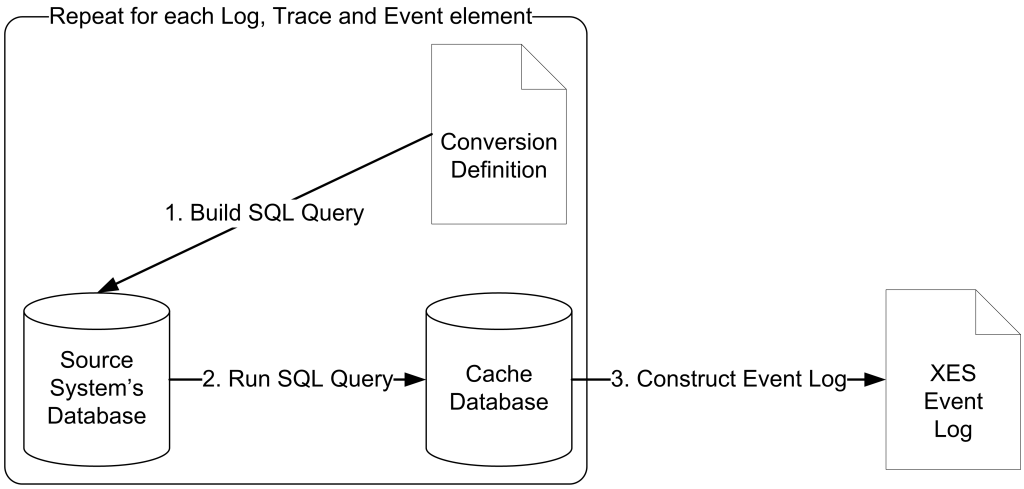


Figure 5.13: The three execution phases of the application.

```
SELECT orderID AS [traceID], timestamp AS [orderAttribute], eventname AS [
    concept_name], eventType AS [lifecycle_transition], users.usergroup AS [
    org_group], users.username AS [org_resource], users.userrole AS [org_role],
    timestamp AS [time_timestamp]
FROM events.csv AS events INNER JOIN users.csv AS users ON users.userID = events.
    userID
WHERE orderID IN (3, 2, 1);
```

Listing 5.1: Example query for an event

entered in the application in Figure 5.9. Each column is given the name of the attribute. The first attribute for instance is the 'concept:name' attribute which will have the value of the 'eventName' column. In this example the attribute values are all single columns but any valid SQL part can be entered.

A special case are those attributes that have child-attributes. The definitions of these child-attributes are included in the same query as their parent. Therefore attributes don't have properties of their own, the properties of the log, trace or event they belong to are used. Internally a record is kept of which columns in the query result represents what attribute.

The 'FROM' part of the query is a combination of the 'from' specification of the conversion definition and the 'links' specified. These are combined together to link all required tables together to extract the attribute values. In this case the event property 'from' was specified as `events.csv AS events`. There was one link defined which was added to the from part of the query.

In the event specification the 'where' property was left empty. In the query a 'WHERE' clause is added nonetheless. Before events are extracted from the data source, the traces are already present in the intermediate database. Only the events for traces that we have extracted should be included. Therefore we automatically select only those events where the specified traceID is in the list of already retrieved traceIDs. In this case we extracted traces with an orderID of 1, 2 and 3 in the previous step. Hence, we automatically only select those events that relate to one of these orders. The event for order 321 is therefore not included in the event log.

5.4.2 Executing the Query

The second execution step is to run the query and store the results in the intermediate database. For each log, trace or event a new record is created in the correct table. The special columns, such as 'traceID' and 'orderAttribute' are added to this record. Each attribute that belongs to the current element is then added to the attributes table of the intermediate database. Each attribute

is given a reference to the element they belong to. Child-attributes are related to the attribute they belong to. Furthermore, the conversion definition is used to specify the key and type of each attribute. An instance of the intermediate database was shown in Figure 4.6.

5.4.3 Constructing the Event Log

Once each query is build, executed and the results are added to the intermediate database the third step can be executed. This step consists of creating an XES event log with the data present in the intermediate database. Listing 5.2 shows part of the resulting event log.

For this step the OpenXES Java library is used to write the event log in the correct format. Before the contents of the database is added to the new event log, other information needs to be added to the header of the event log. The first thing to add to the event log are the extensions used. The extension URL, name and prefix are added in the event log header. In the next step the trace and event globals are detected. Globals are those attributes that are defined for each trace or event. Each attribute that has a value definition is considered to be a global attribute. The resulting attributes are then written to the correct global attributes list. Next the event classifiers as defined in the conversion are added to the event log. All the data so far was present in the conversion definition itself. In the next steps the data stored in the intermediate database is used. First the log attributes are extracted from the intermediate database and added to the event log. Next the traces are extracted. For each trace its attributes and events are retrieved from the intermediate database. The attributes are retrieved in the order as specified by the conversion definition. The trace attributes and the events with their attributes are added to the event log. Then the next trace with its attributes and events is added to the event log. Attributes that have child-attributes are detected during these operations. If one such attribute is encountered its child-attributes are retrieved from the intermediate database and added to that attribute in the event log. This all happens in the same order as the data occurs in the event log. This way the event log can be written in one sequence which is good for the overall performance.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This file has been generated with the OpenXES library. It conforms -->
<!-- to the XML serialization of the XES standard for log storage and -->
<!-- management. -->
<!-- XES standard version: 1.0 -->
<!-- OpenXES library version: 1.0RC7 -->
<!-- OpenXES is available from http://code.deckfour.org/xes/ -->
<log xes.version="1.0" xes.features="arbitrary-depth" openxes.version="1.0RC7"
    xmlns="http://code.deckfour.org/xes">
  <extension name="Lifecycle" prefix="lifecycle" uri="http://code.fluxicon.com/xes/
      lifecycle.xesext"/>
  <extension name="Time" prefix="time" uri="http://code.fluxicon.com/xes/time.
      xesext"/>
  <extension name="Concept" prefix="concept" uri="http://code.fluxicon.com/xes/
      concept.xesext"/>
  <extension name="Semantic" prefix="semantic" uri="http://code.fluxicon.com/xes/
      semantic.xesext"/>
  <extension name="Organizational" prefix="org" uri="http://code.fluxicon.com/xes/
      org.xesext"/>
  <global scope="trace">
    <string key="concept:name" value="UNKNOWN"/>
  </global>
  <global scope="event">
    <string key="concept:name" value="UNKNOWN"/>
    <string key="lifecycle:transition" value="UNKNOWN"/>
    <string key="org:group" value="UNKNOWN"/>
    <string key="org:resource" value="UNKNOWN"/>
    <string key="org:role" value="UNKNOWN"/>
    <date key="time:timestamp" value="1970-01-01T00:00:00.000+01:00"/>
  </global>
  <classifier name="Activity classifier" keys="concept:name lifecycle:transition"/>
  <string key="concept:name" value="LogTest01">
    <date key="Generated_On" value="2010-01-29T10:28:36.000+01:00"/>
    <string key="Generated_By" value="Joos Buijs"/>
  </string>
  <trace>
    <string key="concept:name" value="Order: 1"/>
    <event>
      <string key="org:group" value="Purchase"/>
      <string key="org:resource" value="George"/>
      <string key="lifecycle:transition" value="Start"/>
      <date key="time:timestamp" value="2009-01-01T10:00:00.000+01:00"/>
      <string key="org:role" value="OrderCreator"/>
      <string key="concept:name" value="Create"/>
    </event>
    <event>
      <string key="org:group" value="Purchase"/>
      <string key="org:resource" value="George"/>
      <string key="lifecycle:transition" value="Complete"/>
      <date key="time:timestamp" value="2009-01-01T11:00:00.000+01:00"/>
      <string key="org:role" value="OrderCreator"/>
      <string key="concept:name" value="Create"/>
    </event>
  </trace>
  <trace>
  ...
  </trace>
</log>
```

Listing 5.2: Part of the XES event log that is the result of the conversion.

5.5 Conclusion

This chapter discussed how the solution approach of Chapter 4 was implemented in the application. First, we discussed some of the main decisions made during implementation. Then we discussed the internal structure of the application in more depth. We showed that by applying the model-view-controller pattern the application becomes more flexible. This allows the application to be run without graphical user interface accepting calls from other applications. The graphical user interface was also discussed in detail. It showed how the conversion can be defined by the user. It also demonstrated how expressive the conversion definition is. Next, we showed how the conversion definition is executed. The steps taken to generate the event log were discussed in detail. In the next chapter the application will be tested on two cases from industry.

Chapter 6

Case Studies

To test the application two case studies have been performed. For each case study the source data used in the conversion is explained. The conversion definition is also explained in detail. For each case study multiple conversions have been executed to investigate the performance of different sized event logs. Moreover, the resulting event logs are discussed.

Both case studies are performed on different machines with identical hardware. The computers are manufactured by Dell and are of the 'Precision T5400' model range. They contain two Intel Xeon E5430 processors at 2.66Ghz containing 8 cores in total. They contain 16 GB of memory and run a 64-bit version of Windows. Case study 1 is performed using Windows 7 and case study 2 using Windows Vista.

6.1 Case 1: SAP

This case study uses a data export from part of an SAP system from a large Dutch multinational. The data for this case study is provided by LaQuSo[1].

6.1.1 Source Data

The exported data is stored in a Microsoft Access database with a file size of 1.5 GB. The data is accessed using a JDBC to ODBC bridge driver provided by SUN. This bridge driver then connects to the ODBC connection set up to the Microsoft Access database. The database contains 14 tables, of which we will use 5. These 5 tables with their columns and relationships are shown in Figure 6.1. The number of records in each table is shown in Table 6.1. The same data structure was used in Section 2.1.1 to illustrate some of the event data present in SAP.

The tables and columns shown in Figure 6.1 are related to the procurement process. Please note that the relationships between the tables were added manually. The SAP database does not contain any relationships as these only exist in the application layer. The value 'PK' before a

[1]See http://www.laquso.com

Table 6.1: SAP table record counts.

Table	Number of records
CDHDR	619,262
CDPOS	1,951,148
EKBE	281,745
EKKO	158,436
EKPO	400,99

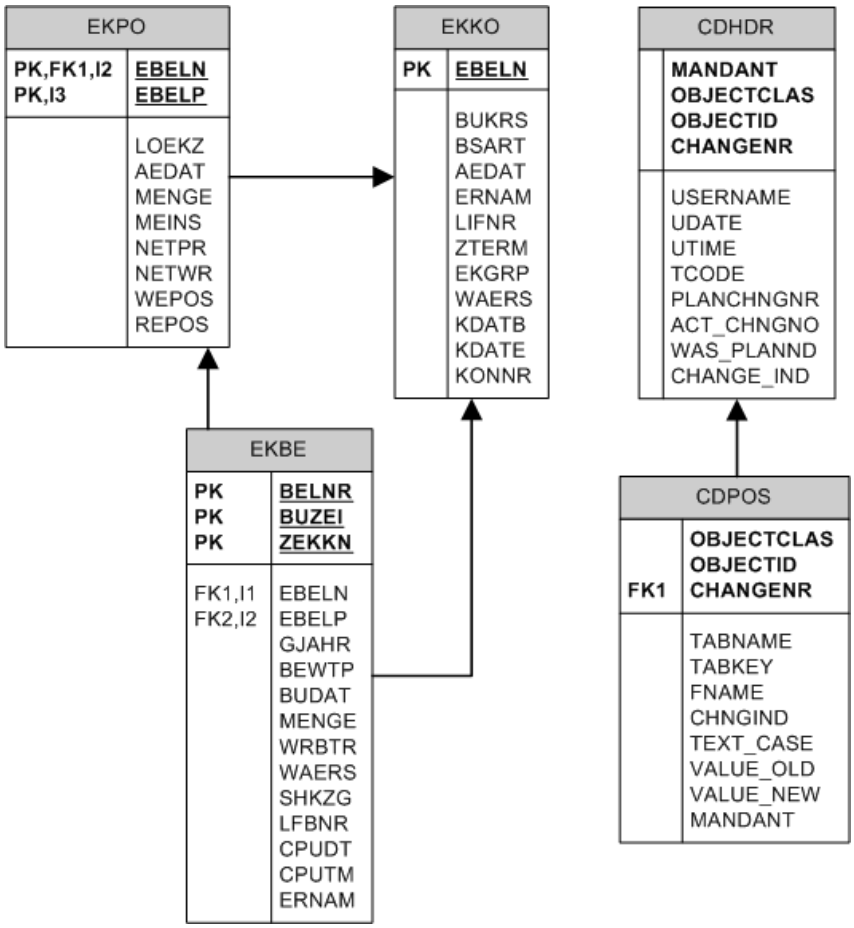


Figure 6.1: SAP source data structure

column name indicates that the column is (part of) the primary key of that table. A primary key is a set of columns that uniquely identifies each record. The column names in bold are mandatory columns for that table. Whenever a column name is preceded by an 'FK' indicator it means that this column refers to (one of) the primary keys of another table.

The table *EKPO* stores order line information. Each order line is related to an order as stored in table *EKKO*. Each order has a unique number stored in the *EBELN* column of the *EKKO* table. Each order line is identifiable by a combination of the order identifier (*EBELN*) and a separate order line identifier stored in the *EBELP* column of the *EKPO* table. The *EBELN* column of the *EKPO* table refers to the order in the *EKKO* table it belongs to. Other attributes included in the *EKPO* table for order lines are the unit (in the field *MEINS*) and the value (*NETWR*) of the order line.

The *EKBE* table stores the history of purchase documents. This table contains information of the handling of goods and invoices related to the orders. The *CDHDR* and *CDPOS* tables store changes made to records in the system. Each record in the *CDPOS* table belongs to a change header in the *CDHDR* table.

6.1.2 Conversion Definition

The conversion definition is based on research conducted within LaQuSo on how process mining can be applied on SAP systems [8]. The entire conversion definition as used in this case study is shown in Appendix D.1.1.

In the conversion we use the order line as our trace instance. Per trace we include particular information elements about the order line. We include the document type which is stored in the *BSART* column of the *EKKO* table. We also store the supplier identifier as stored in the *LIFNR* column of the *EKKO* table. The number of items and the unit in which they are ordered are extracted from the *MENGE* and *MEINS* column from the *EKPO* table. The value of the purchase order is also added to the event log and is taken from the *NETWR* column of the *EKPO* table. We also include the attributes 'GR ind' and 'PG', of which the meaning is not entirely clear.

The traceID is composed of the entire order line identifier. This is a concatenation of the order identifier, stored in the *EBELN* column, and an order line identification number stored in the *EBELP* column. Because many order lines only have a creation event, an extra condition is added. In this conversion we only include order lines which have a record in the change tables. This is indicated in the *where* property of the trace which reads '*EKPO.EBELN&EKPO.EBELP IN (SELECT CDPOS.EKPOKEY FROM CDPOS)*'. The *EKPOKEY* column in the *CDPOS* table is added manually because JDBC does not allow the use of substring functions in this particular case. Furthermore, a link from the *EKPO* table to the *EKBE* table is made requiring a record in the *EKBE* table to exist for the current order line.

The first event we define is the event of creating a new purchase order. This includes the timestamp of creation and the user who created it. The date of creation of a new purchase order is stored in the column *AEDAT* of the *EKKO* table. To ensure that the time is correctly parsed we add a format description to the definition. The user who created the order line is stored in the *ERNAM* column of the *EKKO* table. The event name is set to 'Create PO' and the lifecycle transition to 'complete'.

A change to one of the order lines after creation can be detected by looking at the changes on objects. Changes are recorded in the tables *CDHDR* and *CDPOS*. The *CDHDR* table contains the change header with general information of a change. The *CDPOS* table records the detailed changes made to objects. Each change header can contain multiple detailed changes. For each detailed change the table and value that is changed is stored. Since order lines are stored in the *EKPO* table we search for changes on this table. We select all the change records where the *TABNAME* column of the *CDPOS* table refers to the *EKPO* table. To connect this change to an order line, we use part of the key stored in the *TABKEY* column. Since the substring function is not always allowed by the JDBC driver we added a new column to the *CDPOS* table in the data source. This new column, called *EKPOKEY* stores a substring of the *TABKEY* column. In this case a part of the *TABKEY* is a combination of the *EBELN* and *EBELP* column of the *EKPO* table. The detailed change record in the *CDPOS* table can be connected to the change header via the *CHANGENR* column of both the *CDPOS* and the *CDHDR* tables. The date of the change is stored in the *CDHDR* table in the column *UDATE* and the time in the *UTIME* column. In this case we also added a format description to make sure the date and time is correctly interpreted. In the change header (*CDHDR*) table the user who performed the change is stored in the column *USERNAME*. The event name is set to 'Change line' and the lifecycle transition to 'complete'.

The receipt of goods is also stored in the database. Records in the *EKBE* table where the column *BEWTP* has a value of 'E' indicate receipts of goods. The *EBELN* and *EBELP* columns refer to the order line. The user who entered the receipt in the system is recorded in the *ERNAM* column. The date and time are recorded in the *CPUDT* and *CPUTM* columns respectively. As an attribute the *LFBNR* column is taken, of which the exact meaning is unknown. The event name is set to 'Goods receival' and the lifecycle transition to 'complete'.

Another event recorded in the system is that of receiving an invoice. This event is recorded in the *EKBE* table, if the column *BEWTP* has a value of 'Q'. The username and timestamp are retrieved from the table the same way as for the goods receival event. This event has two attributes. Again the 'lfbnr' attribute taken from the *LFBNR* column. But also the objectkey of the invoice, formed by the *BELNR* and *GJARH* columns, is stored. The event name is set to 'Invoice receipt' and the lifecycle transition to 'complete'.

The *EBKE* table also records other kinds of events. If the value in the *BEWTP* column is equal to 'K' then it indicates an 'account maintenance' event. If the value equals 'L' it is a 'delivery

note' event. If *BEWTP* has a value of 'U' it indicates a 'goods issue' event and when it is equal to 'N' it indicates a 'Subs Deb Log IV' event. The meaning of the last event is not known. For all these events the user who executed the event is stored in the *ERNAM* column. The date and time are stored in the *CPUDT* and *CPUTM* columns respectively. For all these events the lifecycle transition is set to 'complete'.

Figure 6.2 shows a part of the visualization, showing the account maintenance event definition and the *EKBE* database table. The complete visualization is very, very large. This is partly due to the layout algorithm used. This is discussed in more detail in Section 6.3.3.

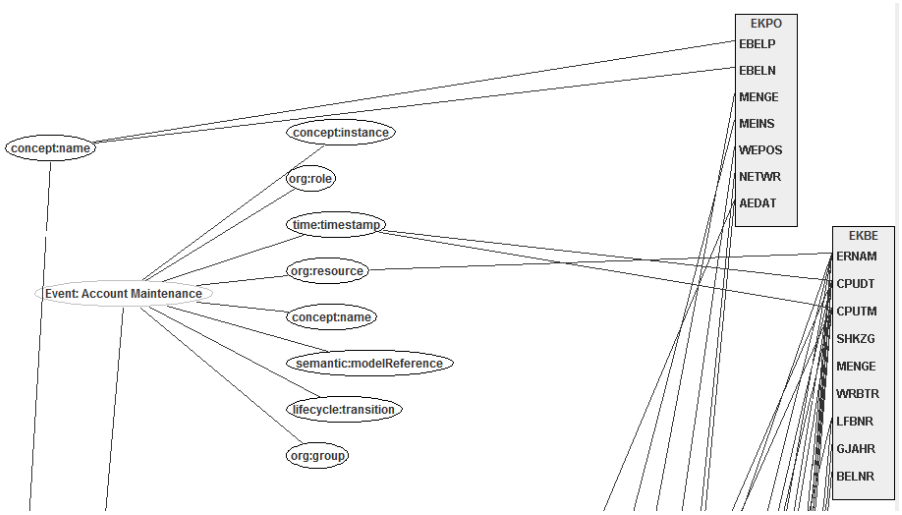


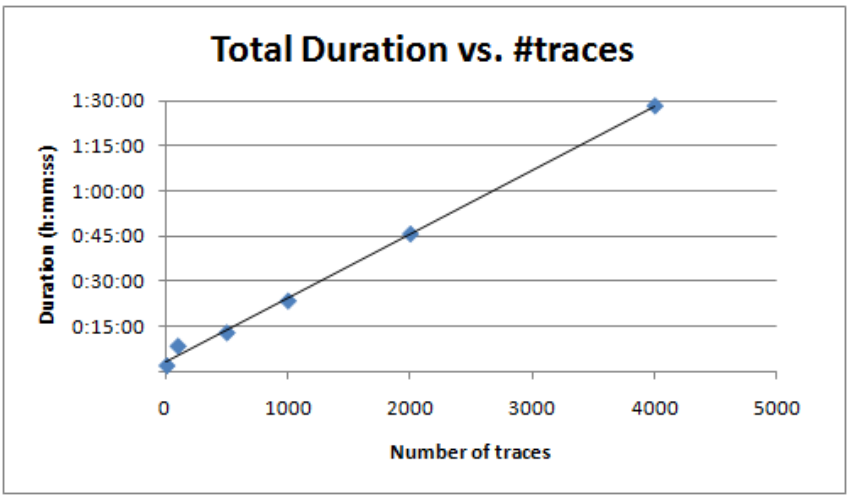
Figure 6.2: Part of the conversion visualization for case study 1.

6.1.3 Conversion Execution

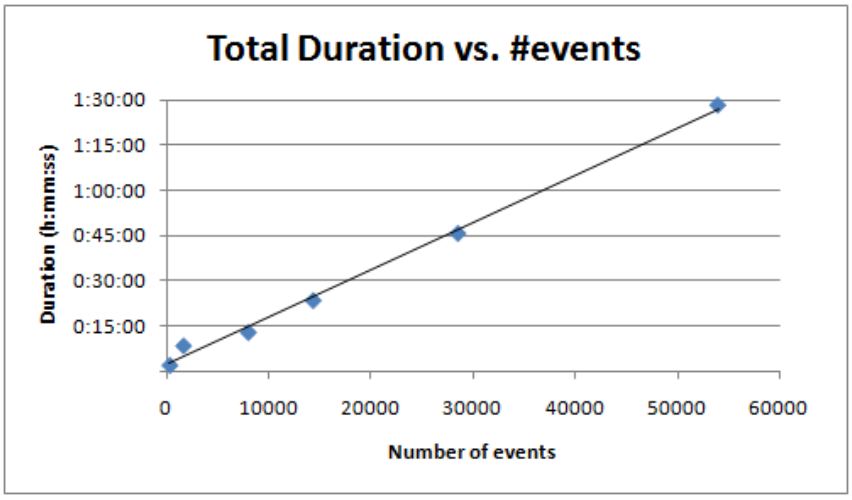
Several conversions were executed to test the performance of the application. The number of traces extracted was changed between different executions. In this section the most important results are discussed. The data on which the following graphs are based is provided in Appendix D.1.2.

Figure 6.3a shows the total duration of conversions for a different number of traces. The graph shows that the time required for the conversion is linearly related to the number of traces included. Further examination shows that the average number of events per trace does not differ much between different conversions. Figure 6.3b confirms that the execution time is linearly related to the number of events included. The time required per event included is shown in Figure 6.3c. This graph shows that on average 10 events per second are processed for larger volumes and that this is constant. It should be noted that conversions with 100 traces or less perform relatively worse than larger conversions. This is due to the fixed time required to set up connections, run queries and write the XES event log.

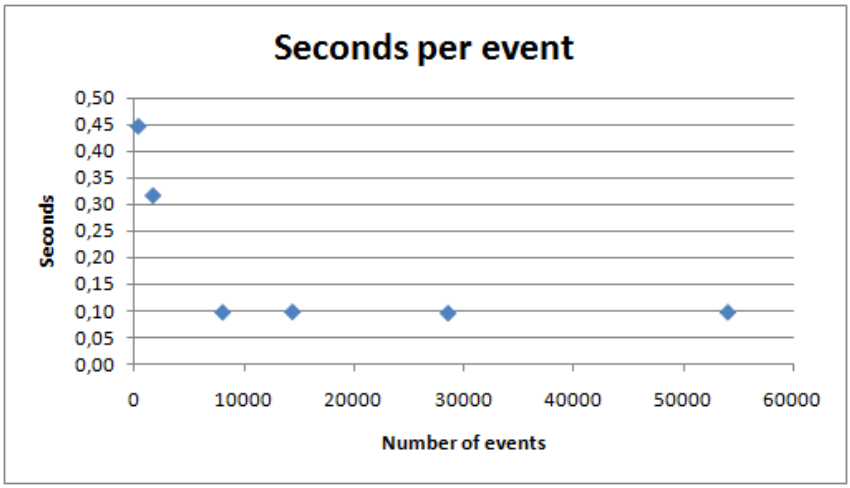
The first conversion executed with 500 traces took 30 minutes while a conversion including 1000 traces only took 23 minutes. To investigate if this was structural or coincidence more conversions were performed. It appeared that it was indeed coincidence since the other two conversions with 500 traces took 13 minutes and 1 second and 13 minutes and 7 seconds respectively. The duration per step for all three conversions with 500 traces is shown in Figure 6.5a. The exact meaning of each step can be derived from Figure 6.4. The graph clearly shows that the first conversion (represented by the blue star symbol) is slower in almost every step than the other two runs. Especially the last step, converting the cache database to the XES file, takes considerably more time for this conversion. Figure 6.5b shows the duration of each step for the three conversions with 1000 traces included. In this graph there is almost no difference between different runs. This indicates that performance in general is consistent between different conversions. Therefore the



(a) Case study 1: Conversion duration versus number of traces.



(b) Case study 1: Conversion duration versus number of events.



(c) Case study 1: Time required per event.

Figure 6.3: Case study 1: Performance charts.

conversion execution with 500 traces that took 30 minutes is left out of the general performance analysis.

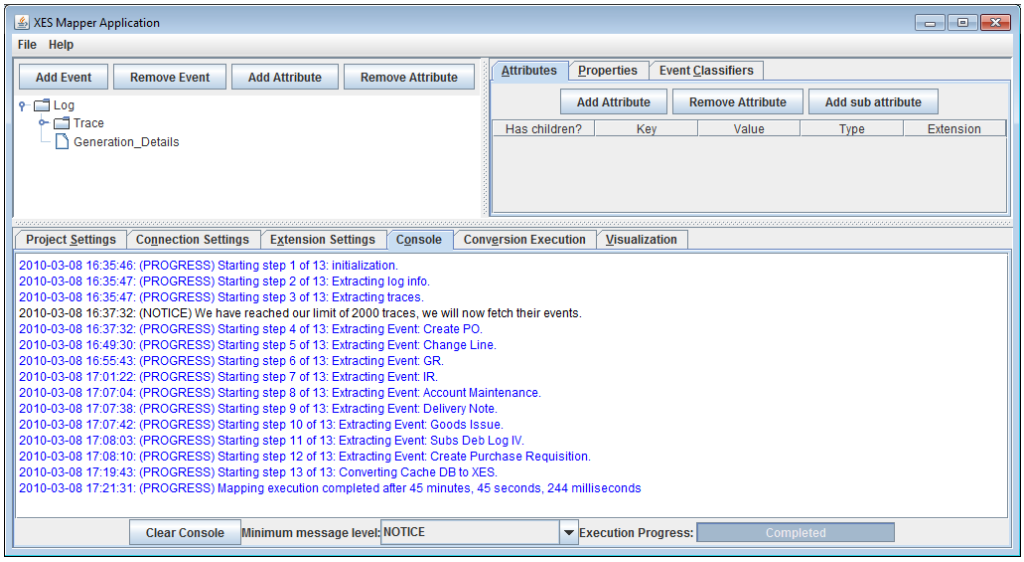


Figure 6.4: Case study 1: Execution console.

The graphs of Figure 6.5a and 6.5b also show that steps 4 and 12 took the most time. Step 4 is the extraction of the event 'Create PO' from the data source. Step 12 is the extraction of the event 'Create Purchase Requisition' from the data source. The event 'Create PO' occurs 2,000 times in the event log with 2,000 traces. The 'Create Purchase Requisition' occurs 1,022 times. Only these two events use a join of the tables `EKPO` and `EKKO`. Therefore we can assume that the additional execution time required by the join in the query causes the extra time required for these steps.

It should be noted that performance is influenced by many factors. For instance the system the conversion is run on and the connection to the data source influence the overall performance. Therefore, conversion executions might perform better or worse than the data provided in this section. The data does show that the performance is linearly related with the number of events included in the conversion. Hence, by performing smaller test runs an estimation can be made about the total run time for a certain number of traces or events.

6.1.4 Event Log

The event log that is the result of the conversion has the characteristics shown in Figure 6.6. The event log contains 14 events per case on average. As can be seen in the top graph of Figure 6.6 there are many traces which only contain three events. Very few traces contain many events, up to 223 events per trace. Eight different event classes are present in the event log and one event type. In total nine different event classes were defined in the conversion. The event 'Delivery Note' does not appear in the event log.

The process model discovered using the Heuristic miner is shown in Figure 6.7. This process model displays each activity in the event log. The arrows between activities indicate the order in which they are executed for the majority of traces in the event log. The process model clearly shows that each process starts with the creation of a purchase order. The rest of the process model is not very structured and many relations between activities exist. What is remarkable is that the 'Create purchase requisition' event always occurs later than the 'Create PO' event. This could indicate that the database field used for the timestamp of the 'Create purchase requisition' event is incorrect and does not contain the creation date. Another thing that is remarkable from this

Table 6.2: Case study 1: Event occurrences.

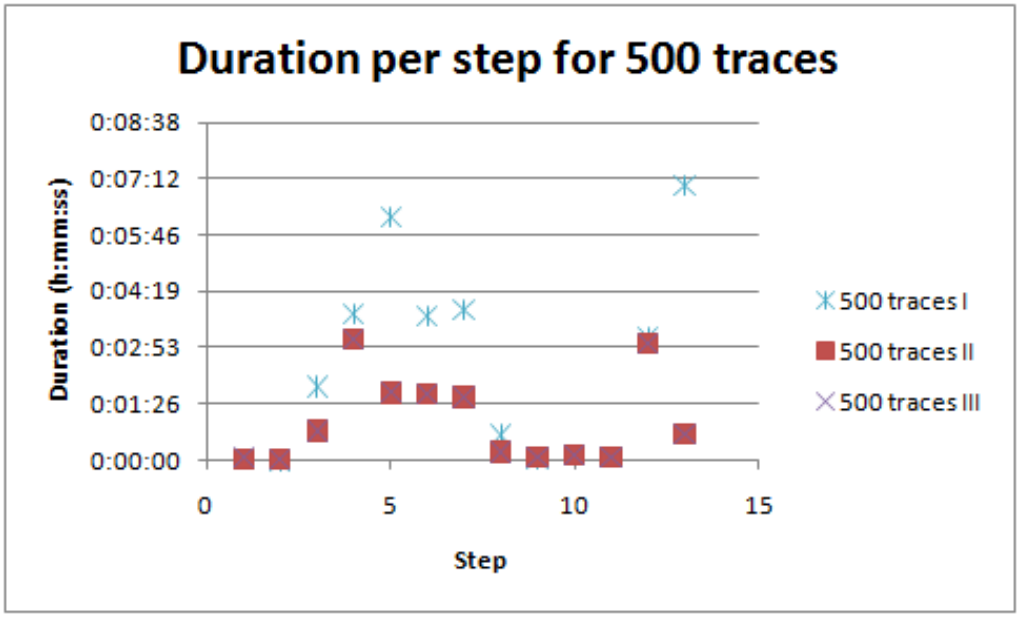
Class	Occurrences (absolute)	Occurrences (relative)
Change Line+complete	15,883	55.69%
IR+complete	4,328	15.18%
GR+complete	4,165	14.60%
Create PO+complete	2,000	7.01%
Create Purchase Requisition+complete	1,022	3.58%
Account Maintenance+complete	987	3.46%
Subs Deb Log IV+complete	79	0.28%
Goods Issue+complete	56	0.20%

process model is the fact that a 'Subs Deb Log IV' is always followed by 'Account Maintenance'. Furthermore, the 'self-loops', where events are executed multiple times, is remarkable.

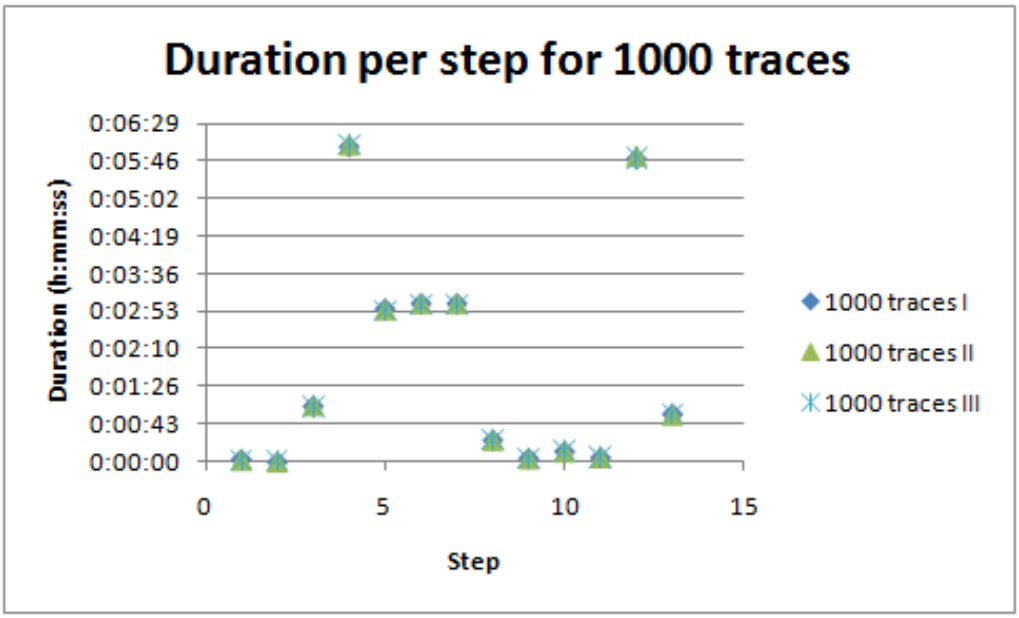
6.1.5 Alternative Conversion Method

This data has been converted to an event log before, without the use of this application. Several separate queries were created to extract certain information from the data source. This information was stored in an intermediate Microsoft Access database. This intermediate database contained the traces and events and their attributes. The intermediate database format was that used by the 'MS Access Database' conversion plug-in of ProM Import. This ProM Import plug-in was modified to use the sequential log writer instead of the buffered log writer to increase performance. On a database containing over 187.000 traces and 627.000 events in total the plug-in ran for about 10 minutes.

The main disadvantage of this method was that much knowledge of database systems was required. Complex queries needed to be written efficiently. Furthermore, the intermediate database needed to be set up by hand. The ProM Import plug-in also needed to be adjusted in order to successfully execute the conversion. Another disadvantage of this method was that the conversion to the intermediate database format was laborious. Getting the conversion chain to produce an MXML file took multiple days because of many technicalities encountered. Over 30 queries were needed to extract the traces, the events and their attributes from the data source. Each of these queries needed to be started by hand, in the correct order, to fill the intermediate database. If an error was discovered in one of the queries, the intermediate database would be emptied and each of the queries was run again. The time needed to run these queries is not included in the runtime of the ProM Import conversion plug-in and was between 10 and 20 minutes.



(a) Case study 1: Time per execution step for 500 traces.



(b) Case study 1: Time per execution step for 1000 traces.

Figure 6.5: Case study 1: execution times per step.

Figure 6.6: Case study 1: ProM log summary overview

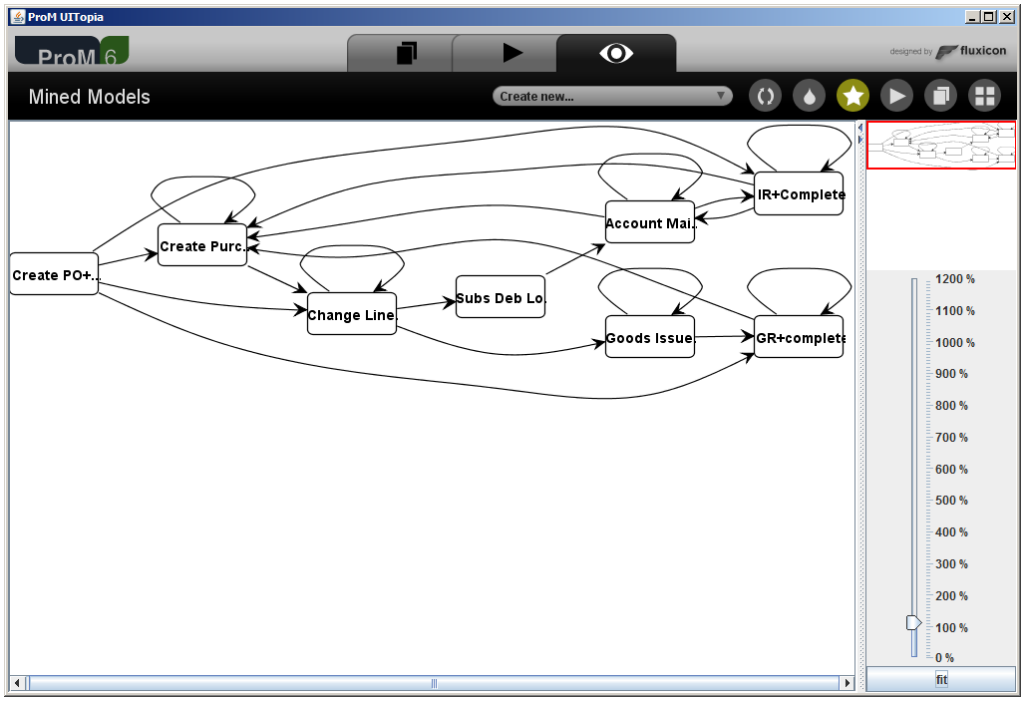


Figure 6.7: Case study 1 Heuristic miner model

6.2 Case 2: Custom System

The second case study uses data exported from a custom-built workflow management system. The system serves as a central administrative system. Its goal is to trigger other systems or users to perform certain activities on cases. It is built and used by a large semi-governmental company in the Netherlands.

6.2.1 Source Data

The database export received from the company consisted of several text files in comma separated value (csv) format. The separator was the '|'-character. The export contained information on cases from the year 2008. From the thirteen files provided two were used in our conversion. Characteristics of the tables used are shown in Table 6.3. The first table contains the history of activities performed on cases. The second table contains more detailed information on the defined tasks. This table is used to get the name of the activity performed, instead of a number.

The data is accessed using an ODBC to csv driver present in Windows by default. This driver allows access to csv-files as if they were tables using the ODBC interface. The default JDBC to ODBC bridge provided by SUN was used to connect to this ODBC driver.

6.2.2 Conversion Definition

The conversion definition on the data source was rather straight forward. Detailed information of the conversion definition is shown in Appendix D.2.1. Two tables were used from the data source. The first, the history table, contained essentially all information required. It contained the case identifier that indicated for which case the activity was performed. It also contained an activity identifier. This referred to a record in another table from which the corresponding activity name was retrieved. The history table also contained information on which user performed a certain activity. The table also contained information on the start and completion time of a certain event. This meant that two event definitions were required; one for indicating the start of a certain event and another to indicate the completion of that event. The conversion definition is visualized in Figure 6.8.

6.2.3 Conversion Execution

Multiple conversion executions have been performed using the same definition. By increasing the number of traces included in the event log the scalability of the XESMa application has been investigated. Since a dedicated conversion had been implemented before, performance is compared to this implementation. The dedicated conversion implementation will be discussed in more detail in Section 6.2.5. The comparison between the dedicated implementation and the XESMa application is shown in Figure 6.9. The dedicated implementation is designed to only walk through the data source once, extract information from the source and store it in memory. XESMa will perform three queries on the data source, hence requiring to go through the data three times. The first time it extracts the traces, the second time all the events of type 'start' and the third time those of type 'complete'.

What is surprising is that XESMa is significantly faster for small event logs. This can be seen in more detail in Table 6.4. The dedicated application always needs a bit more than one hour and

Table 6.3: Data characteristics for case study 2.

Table	**Contents**	**Number of records**	**File size**
History	Case history	19,223,294	2.14 GB
Activity	Task details	811	45 KB

Table 6.4: Case study 2: Performance comparison between dedicated implementation and XESMa.

Traces	Dedicated	XESMa
10	1:23:43	0:26:33
20	1:22:49	0:30:39
50	1:21:46	0:43:13
100	1:21:36	1:01:37
200	1:21:15	1:40:11
500	1:22:14	3:35:52

20 minutes to create an event log, regardless of the size. XESMa is able to create an event log in under one hour when less than 100 traces are included. If 500 traces are included execution time has grown to three and a half hours. This could be caused by the comma-separated-values ODBC driver. It could be that some kind of sorting is automatically performed on the source text file. This would allow the ODBC driver to only search part of the text file.

Another difference between XESMa and the dedicated implementation is that XESMa does not keep results in memory. The dedicated implementation stores everything in memory while parsing the text file. Once the text file is completely gone through it writes the memory contents to an XES event log. This works if the event log fits in memory, if this is not the case then the dedicated implementation will not work.

The relation between the number of traces and the time needed to execute the conversion is linear. When the conversion duration is compared to the number of events included in the event log the graph of Figure 6.10 is the result.

Dividing the total time required for the execution by the number of events in the event log results in the graph of Figure 6.11.

The overall performance of XESMa is about a factor 10 worse than what was achieved in case study 1. This can be explained by the fact that for case study 1 the data was stored in a local database with indexes to improve searching. For this case study the data source was a plain text file over 2 GB in size. Furthermore, XESMa needed to go through this file three times to extract all the information required. This explains the increase in time required to execute the conversion. Nonetheless, the time required by XESMa is still linear with relation to the number of traces and events included.

6.2.4 Event log

Figure 6.12 shows the log summary after the event log has been loaded into the development version of ProM 6. It shows that the event log contains 500 traces and 13,342 events. There are 338 different event classes and two different event types. Each trace contains at least two events, on average 26 events and at most 492 events. Process models have been discovered using the Alpha algorithm and Fuzzy miner plug-in implemented in the development version of ProM 6. The resulting process models are shown in Figure 6.13. Both process models show a rather unstructured process with a lot of different connections between activities. One of the characteristics of this event log is that it contains records of several small sub-processes. This means that there are many different relationships between events. Therefore, it is hard to create a small and correct process model.

6.2.5 Alternative Conversion Method

As discussed in Section 6.2.3, this data source has been converted to an event log before. The conversion was implemented in Java for this particular source format. The Java application goes through the entire text file, processing each line. From each line it extracts event information if

the trace is included in the event log. The complete event log is kept in memory until the text file is completely parsed. It then writes the traces and events to an event log.

The disadvantage of this method is that it was custom build for this particular data source. Furthermore, if the notion of a trace was changed from a case to a customer, a large part of the code needed to be changed. Another disadvantage is that it stores the event log information extracted from the text file in memory. If the event log does not fit in memory any more the application will crash.

An advantage of the specific implementation is that the time required is more or less constant, regardless of the number of traces included, as shown in Figure 6.9

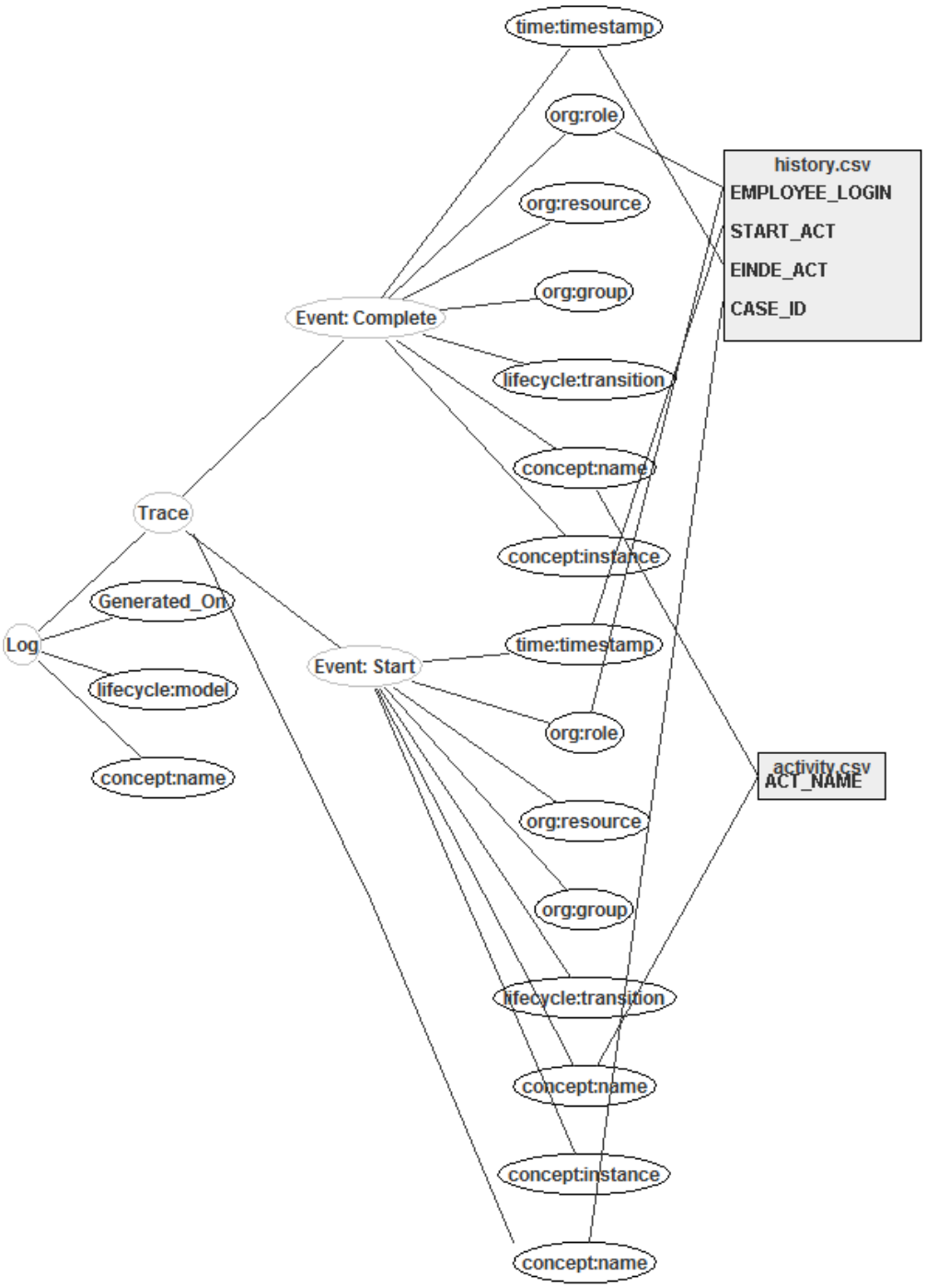


Figure 6.8: Conversion visualization for case study 2.

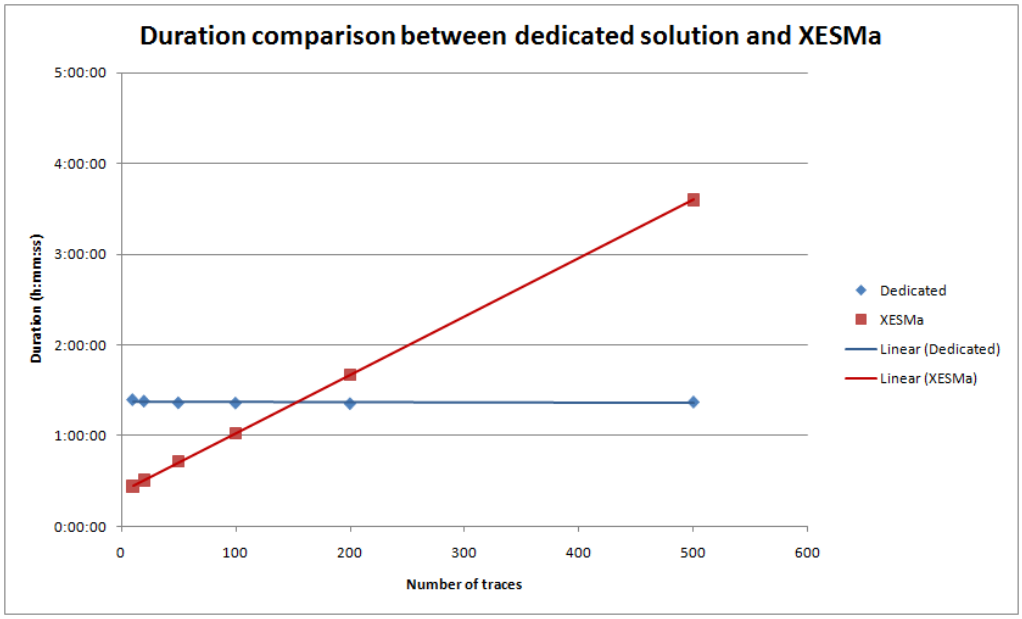


Figure 6.9: Case study 2: Comparison between the dedicated implementation and XESMa.

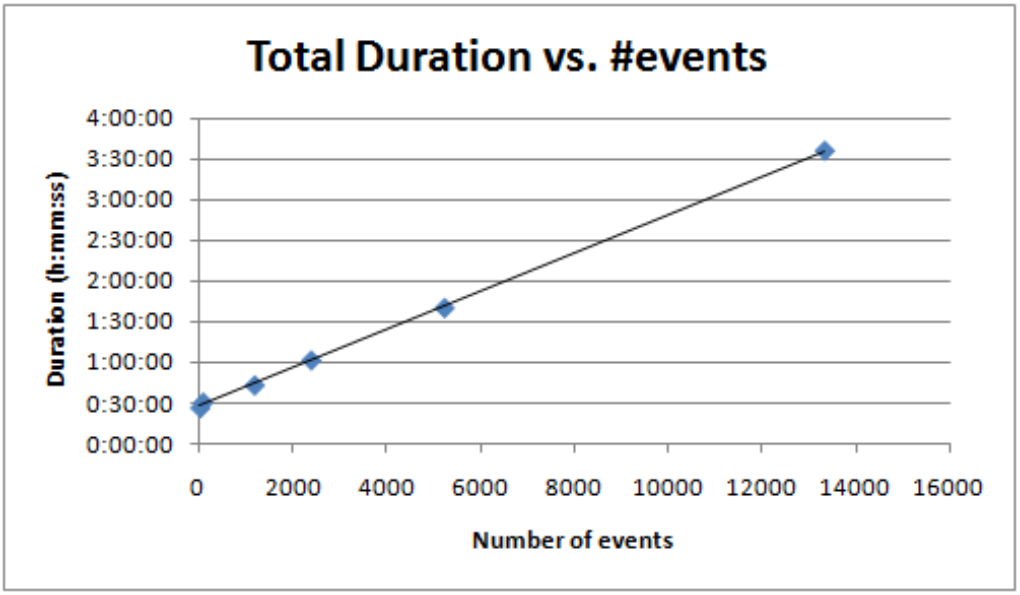


Figure 6.10: Case study 2: Conversion duration versus number of events.

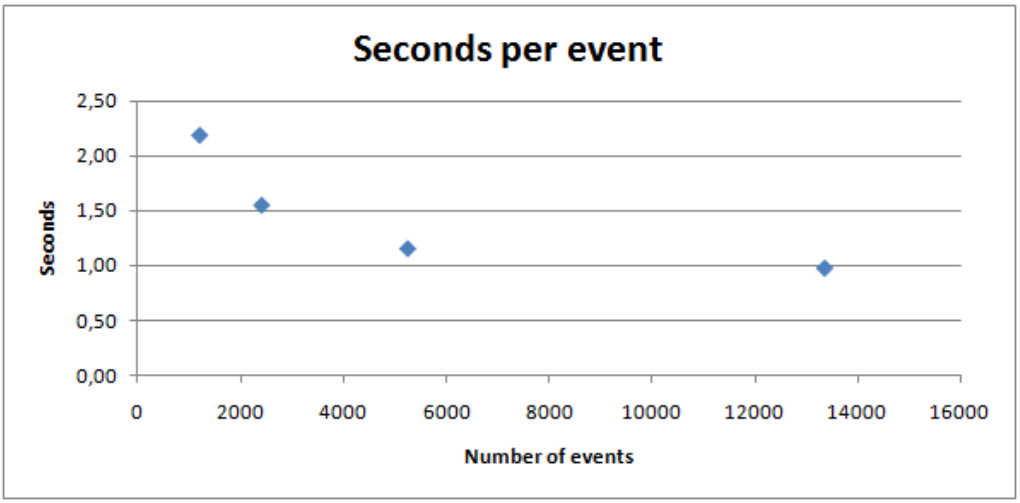
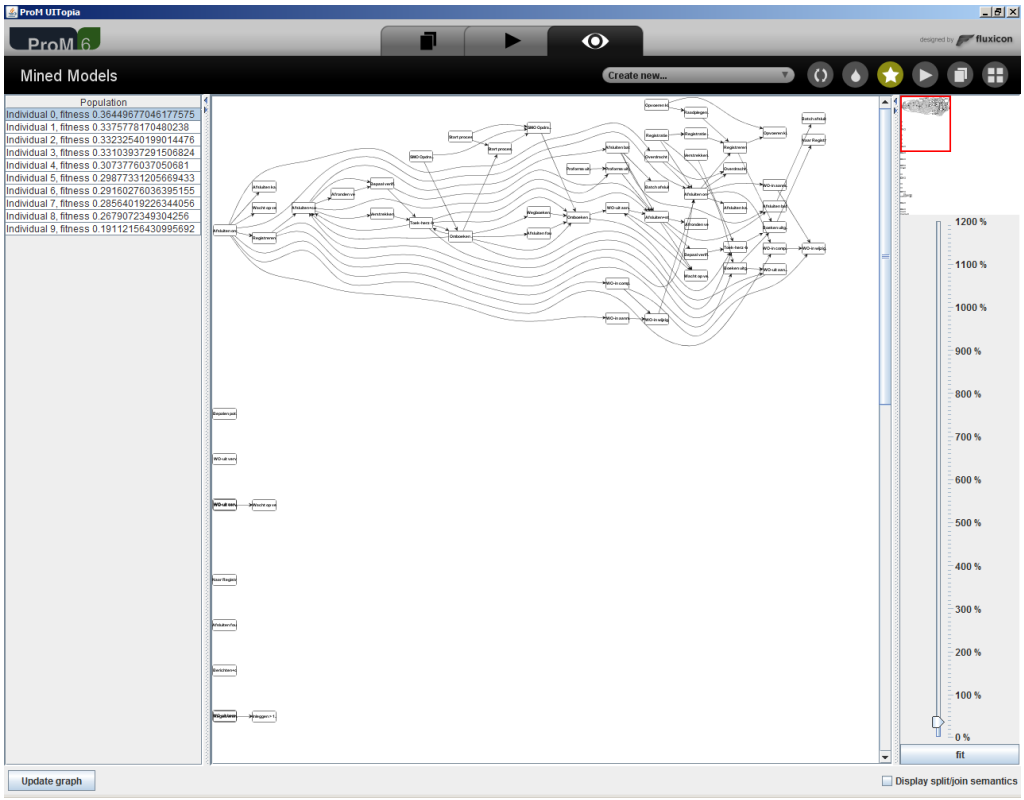
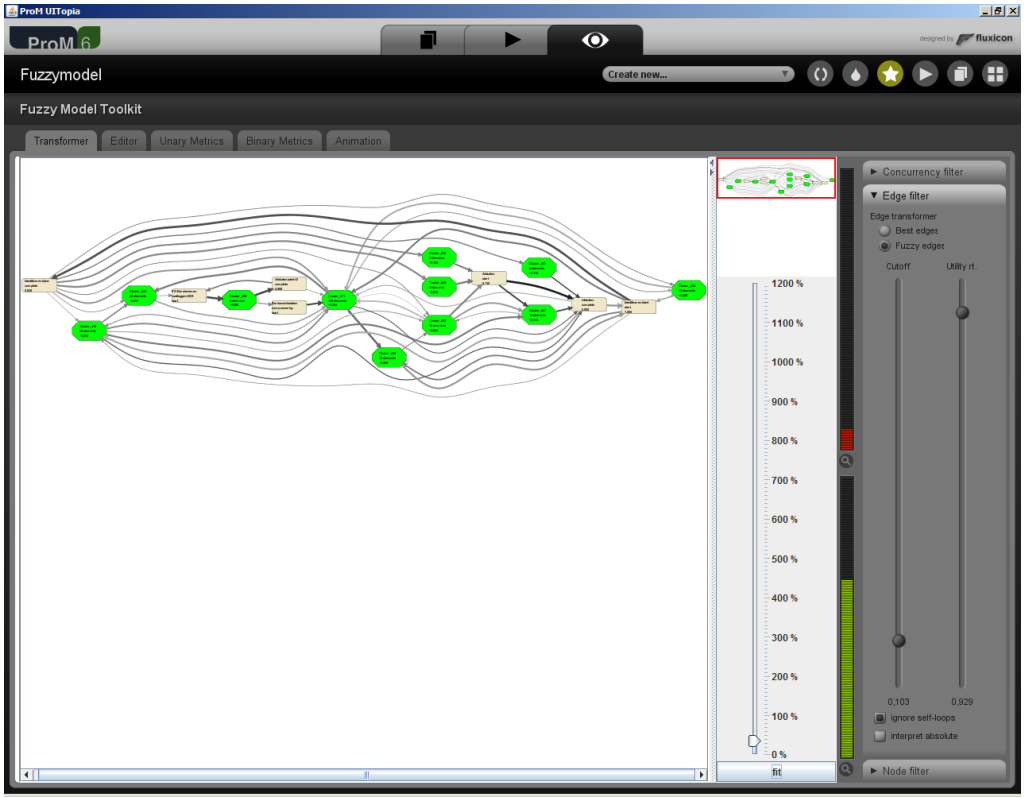


Figure 6.11: Case study 2: Time required per event.

Figure 6.12: Log summary in ProM 6.



(a) Process model using the Alpha algorithm.



(b) Process model using the Fuzzy miner.

Figure 6.13: Process models mined from the event log of case study 2 using two different plug-ins.

6.3 Limitations Discovered

While performing the case studies some limitations of the application or the technology used were discovered. These are discussed in this section.

6.3.1 Using SQL Functions

During testing of the application a problem was discovered when using SQL functions in the conversion definition. Examples of such functions are substring and mathematic functions. Usage of these functions in combination with certain other features in the query failed. The problem probably lies somewhere in the JDBC-ODBC bridge driver or in the ODBC drivers used. The error occurred using both the csv-file and MS Access database ODBC drivers for both our case studies. Both these drivers are provided by Microsoft with the Windows operating system. No other target databases have been tested so far. The same query that can not be run through the JDBC and ODBC connection could be run without problems directly in the MS Access database. This indicates that the error probably lies somewhere in one of the drivers used. The result of this error is that the conversion definition is less powerful. More complex operations such as taking only part of a string or conditional values are no longer possible.

Listing 6.1 shows a simplified version of the query of Listing 5.1. Two alternatives for the value of the 'test_attribute' are used and both can be run without errors on the data source. However, if we run any of the queries in Listing 6.2 we get the error '`[Microsoft][ODBC Text Driver] You tried to execute a query that does not include the specified expression 'TRACEID' as part of an aggregate function.`'. The first query in Listing 6.2 tries to extract a substring of the eventname column using the *MID()* function. The second query tries to include the current timestamp using the *NOW()* function of SQL. The third query also tries to use a part of the value of the eventname column but now using the official JDBC syntax for functions[2]. The fourth query uses the official JDBC function syntax to include the current timestamp. All four queries result in the same error and fail to run. This also indicates that whether or not the special JDBC syntax for calling functions is used, it will always fail.

6.3.2 Running on 64-bit Windows

When the application is run on a 64-bit version of Windows it might be that the application has to be run in 32-bit mode. This can be the case if there are no 64-bit JDBC or ODBC drivers available for the source database used. In both case studies we needed an ODBC driver provided by Microsoft. Neither of these drivers was available in a 64-bit version[3]. If the application is run in 64-bit mode then it can only use 64-bit ODBC drivers. Since no such drivers are available, the application needs to be started using a 32-bit Java Virtual Machine.

The consequence of having to run in 32-bit mode instead of 64-bit mode is that the application can not access more than 4 GB of memory. This means that for larger conversions the data can not be kept in memory. This does however prove the benefit of the intermediate database. Because

[2] See http://java.sun.com/developer/onlineTraining/Database/JDBC20Intro/JDBC20.html#JDBC2012
[3] See http://support.microsoft.com/kb/942976

```
SELECT orderID AS [traceID], timestamp AS [orderAttribute], eventname AS [
    concept_name], 'fixed value' AS [test_attribute] FROM events.csv AS events
    WHERE orderID IN (3, 2, 1);

SELECT orderID AS [traceID], timestamp AS [orderAttribute], eventname AS [
    concept_name], eventtype AS [test_attribute] FROM events.csv AS events WHERE
    orderID IN (3, 2, 1);
```

Listing 6.1: Examples of queries that can be run

```
SELECT orderID AS [traceID], timestamp AS [orderAttribute], eventname AS [
    concept_name], MID(eventname,2) AS [test_attribute] FROM events.csv AS events
    WHERE orderID IN (3, 2, 1);

SELECT orderID AS [traceID], timestamp AS [orderAttribute], eventname AS [
    concept_name], NOW() AS [test_attribute] FROM events.csv AS events WHERE
    orderID IN (3, 2, 1);

SELECT orderID AS [traceID], timestamp AS [orderAttribute], eventname AS [
    concept_name], {fn MID(eventname,2)} AS [test_attribute] FROM events.csv AS
    events WHERE orderID IN (3, 2, 1);

SELECT orderID AS [traceID], timestamp AS [orderAttribute], eventname AS [
    concept_name], {fn NOW()} AS [test_attribute] FROM events.csv AS events WHERE
    orderID IN (3, 2, 1);
```

Listing 6.2: Examples of queries that fail to execute

this database is used to store the event log, memory usage is reduced. The alternative would be to load the entire event log into memory before writing it to an XML file.

6.3.3 Visualization Readability

With respect to the visualization more experiments should be done on the way of visualization and on the type of automatic layout to apply. At the moment a tree layout is applied that places the root at the left of the image and works to the right from there. Unfortunately, the connections from the attributes to the columns in the tables make the visualization rather complex to read. The automatic layout algorithm does not take these connections into account. As a result the connections cross each other and other shapes in the visualization.

6.4 Conclusion

In this chapter the application has been applied to two cases from industry. For both cases a conversion could successfully be defined and executed. However, a number of issues were discovered. Most of these issues are of a technical nature and can not be resolved within the application. Despite these limitations, the application demonstrates that it is possible to create a generic application where a non-expert user can define a conversion and generate an event log. It also meets the requirements stated in Sections 1.3 and 4.1. The application is able to connect to the data sources of both case studies. Furthermore, the application is expressive enough for both conversion definitions to be defined. The conversion definition is also easily adjusted when for instance the trace instance is changed to another business object.

Chapter 7

Conclusions

Within this master thesis we analyzed the extraction of event logs from non-event-based data sources. Different conversion aspects of converting generic data to an event log format are discussed. The goal of this master project was defined as follows:

> ***Goal of the project**: Create an application prototype that helps a business analyst to define and execute a conversion from a data source in table format to the event log format XES requiring as little programming as possible.*

Before investigating the problem in more detail, we presented the reader with an introduction into process-aware information systems. We showed that event related data is present in many information systems, including for example SAP. We also discussed event logs in general and their most important properties. Two specific event log formats, MXML and XES, were also discussed. Then, we briefly explained process mining which analyses event logs.

The first contribution we made was a discussion on important aspects to consider when defining a conversion of data to an event log format. We discussed aspects of the selection of different event log elements. We also discussed the frequently recurring problem of convergence and divergence encountered when defining a conversion. These aspects should be taken into account in every process mining project.

Our major contribution is the XES Mapper (XESMa) application prototype. This application provides a generic way for converting data from a data source to an event log. XESMa is designed to be easy to use. One of the key strengths of XESMa is that no programming is required. The entire conversion can be defined through the graphical user interface. XESMa divides the conversion information required into small parts making it easier to define. For more complex conversions SQL knowledge might be required for using XESMa. Since SQL is a declarative language it is easier to learn than programming.

To demonstrate the applicability of our proposed solution two case studies have been performed on data from different systems. The first case study was performed on data from an SAP system. This case study showed that a conversion definition could easily be defined using different tables and columns from the data source. The other case study showed that data exported from a custom system can also be converted to an event log by our application. For both case studies the performance was also investigated and shown to be linear in time with the size of the event log.

We believe that we have shown and supported a generic approach to extract event related data from a data source without the need to program.

7.1 Limitations and Future Work

Of course, applications are never finished and the application itself can be improved in many ways. My main recommendation is to take the XESMa application prototype and make a production ready version. The overall architecture of the prototype is correct but specific implementation

details can be improved. The problems discovered in Section 6.3 should also be investigated further and solved if possible. Furthermore, performance should be improved in order to run XESMa in production environments. This might include an investigation of several (commercial) JDBC drivers and selecting those that perform as desired.

Another main improvement would be to investigate and implement a better way to visualize the conversion definition. Especially visualizing larger conversion definitions can be improved. Within this project there was not enough time to thoroughly test and investigate different visualization settings. A good visualization is critical to the usability of the overall application. Furthermore, it allows a conversion definition to be 'printed' and discussed with business experts in a clear and concise way.

The application can also be extended in several directions. A first extension could be to implement an editor to guide the user in defining attribute values. This would reduce the number of errors that can be made. Furthermore, it provides more guidance to the user and therefore makes the application more user friendly. Another extension could be to implement certain filters that can be applied to the data. An example of such a filter is the anonymization of event logs. The names of users could for instance be replaced by numbers or other strings to remove any link to real users. Another way to extend the application would be to implement suggestions or recommendations for the user. Divergence and convergence between traces and events could be detected for instance. Notification of this to the user could prevent errors in the analysis phase. Another example is when a timestamp is found somewhere in the database. Then it is likely that the timestamp is related to an event that can be included in the event log. By notifying the user of these discoveries unknown events can be discovered and included in the event log.

Besides improving the application itself, further investigation into specific conversion definitions should be conducted. Especially extracting event data from systems such as SAP is laborious and much knowledge of the system is required. A collection of 'reference conversions' should be created. This would prevent inventing the wheel many times over. Even though each SAP system and each process mining project is different, a collection of 'reference conversions' could drastically improve the speed and quality of the event log extraction.

As discussed in Chapter 3 there are important considerations to make when extracting an event log from a data source. These conversion aspects should be extended and formatted as a general guide for extracting event logs. This would provide more detailed guidance in defining event log conversions. Especially for people new to the field of process mining such a guide would help to quickly define an event log conversion and apply process mining on the resulting event log.

Bibliography

[1] W. van der Aalst, B. van Dongen, C. Günther, R. Mans, A. de Medeiros, A. Rozinat, V. Rubin, M. Song, H. Verbeek, and A. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. pages 484–494. 2007. 16

[2] W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems (Cooperative Information Systems)*. The MIT Press, January 2002. 7

[3] W.M.P. van der Aalst, V. Rubin, H. Verbeek, B.F. van Dongen, E. Kindler, and C.W. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 2009. 2, 16

[4] A. K. Alves de Medeiros, W.M.P. Van der Aalst, and C. Pedrinaci. Semantic process mining tools: Core building blocks. In *16th European Conference on Information Systems*, Galway, Ireland, June 2008. 15, 23

[5] A.K. Alves De Medeiros and W.M.P. Aalst. Process mining towards semantics. pages 35–80, 2009. 23

[6] M. Bozkaya, J.M.A.M. Gabriels, and J.M.E.M. van der Werf. Process Diagnostics: A Method Based on Process Mining. *International Conference on Information, Process and Knowledge Management*, 0:22–27, 2009. 22

[7] B. Bruegge and A.H. Dutoit. *Object-Oriented Software Engineering: Using UML, Patterns and Java, Second Edition*. Prentice Hall, September 2003. 43

[8] J.C.A.M. Buijs. SAP Procurement Conversion to MXML. Technical report, LaQuSo (Laboratory for Quality Software), 2009. 60

[9] B.F. van Dongen and W.M.P. van der Aalst. A Meta Model for Process Mining Data. In *Conference on Advanced Information Systems Engineering*, volume 161, Porto, Portugal, 2005. 10, 11, 83

[10] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM Framework: A New Era in Process Mining Tool Support. pages 444–454. 2005. 16

[11] M. Dumas, W.M.P van der Aalst, and A.H.M ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology*, chapter 1, pages 3–20. Wiley-Interscience, Hoboken, NJ, USA, 2005. 7, 8

[12] Ronald Fagin, Laura M. Haas, Mauricio A. Hernández, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema Mapping Creation and Data Exchange. In *Conceptual Modeling: Foundations and Applications*, pages 198–236, 2009. 19

[13] M. van Giessel. Process Mining in SAP R/3. Master's thesis, Eindhoven University of Technology, 2004. 2, 8

[14] C. Günther. *Process Mining in Flexible Environments*. PhD thesis, Eindhoven University of Technology, 2009. 10, 16, 18

[15] C.W. Günther. *XES Standard Definition*. Fluxicon Process Laboratories, November 2009. 13, 14

[16] C.W. Günther and W.M.P. van der Aalst. A Generic Import Framework for Process Event Logs. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops, Workshop on Business Process Intelligence (BPI 2006)*, volume 4103 of *Lecture Notes in Computer Science*, pages 81–92. Springer-Verlag, Berlin, 2006. 18

[17] C.W. Günther and W.M.P. van der Aalst. Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In *BPM*, pages 328–343, 2007. 16

[18] J. Ingvaldsen. *Ontology Driven Business Process Intelligence*. 18, 23

[19] J. Ingvaldsen and J. Gulla. Preprocessing Support for Large Scale Process Mining of SAP Transactions. pages 30–41. 2008. 2, 8, 18, 22

[20] M. Jans, N. Lybaert, and K. Vanhoof. Business Process Mining for Internal Fraud Risk Reduction: Results of a Case Study. 2009. 1

[21] A. Rozinat, I.S.M. De Jong, C.W. Günther, and W.M.P. van der Aalst. Process mining applied to the test process of wafer scanners in ASML. *Trans. Sys. Man Cyber Part C*, 39(4):474–479, 2009. 2

[22] A. Rozinat, S. Zickler, M. Veloso, W.M.P. van der Aalst, and C. McMillen. Analyzing Multi-agent Activity Logs Using Process Mining Techniques. In *Distributed Autonomous Robotic System 8*, page 251. Springer, 2009. 2

[23] I.E.A. Segers. Deloitte Enterprise Risk Services: Investigating the application of process mining for auditing purpose. Master's thesis, Eindhoven University of Technology, 2007. 1, 2, 24, 25

[24] K. van Uden. Extracting User Profiles with Process Mining at Philips Medical Systems. Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2008. 22

[25] Panos Vassiliadis, Alkis Simitsis, Panos Georgantas, Manolis Terrovitis, and Spiros Skiadopoulos. A generic and customizable framework for the design of ETL scenarios. *Inf. Syst.*, 30(7):492–525, 2005. 20

[26] Wikipedia. List of ERP software packages — Wikipedia, The Free Encyclopedia, 2010. [Online; accessed 6-January-2010]. 7

Appendix A

Glossary

Activity	An action or task that can be executed in a process for a process instance. Activities are recorded in events.
Application, the	The prototype application that will be the result of the requirements discussed in this document. See also 'XES Mapper Application'.
Conversion	A definition or execution of a definition on how to extract information from the data source in order to construct an event log.
Data source	Any source containing data that we can use to create our event log. In our case data sources are most likely relational databases.
Event	Recording of an activity performed by a resource on a certain time for a specific process instance.
Event Log	A special type of log which contains a recording of a set of events related to process instances.
GUI	Abbreviation for 'Graphical User Interface', which is the presentation and interaction layer of an application with the user.
General Mapping Item	A generalizing name for the conversion items log, trace, event and attributes. See Figure 4.1 on page 33 for the class-diagram definition of this structure.
Log	A recording of things that happen. Examples are event logs, error logs and access logs.
Mapping	A conversion definition from one data source to another. In the context of this document we mean from the data source(s) to an XES event log.
MXML	A general event log format, superseded by the XES format (see XES).
OpenXES	A Java library to read, modify, and write XES event logs and XES extensions.
Process instance	An object that passes through a process. Examples are cases, patients, purchase orders, or machines. In the XES event log this is represented by the trace element.
Process mining	Analyzing a business process based on an event log, see `http://www.processmining.org`.
ProM	An application to apply several process mining techniques on an event log, see `http://www.processmining.org`.

Continued on next page

Record	One row in a file or database table.
Relation	Tables in a database might be related to each other. One field in a table refers to a key field in another table. An example could be an order table with a relation to the user table. The order table contains information about the different orders and the user table about the different users. The relationship could for instance indicate which user created the order.
Resource	Any actor that can execute an activity, for example humans or a web service.
Timestamp	A date and time.
Trace	An ordered set of events related to a single process instance.
XES	XES is a flexible event log meta-model, see `http://code.deckfour.org/xes`.
XESMa	The abbreviation for the XES Mapper Application developed in this thesis.
XES attribute	An attribute name and value related to an XES log, XES trace, XES event or even another XES attribute.
XES event log	An event log in the XES format.
XES element	One of the elements present in XES: log, trace, event or attribute.
XES event	An event in the XES log.
XES Extension	By default the XES meta-model has no attributes attached to any of the XES elements. Extensions can be used to define attributes for an XES log, XES trace, XES event or even to other XES attributes. The attributes defined by an extension have a clear semantic meaning. Examples of attributes are event name, name of the originator, and execution timestamp in the event log.
XES log	A set of traces in the XES log.
XES Mapper Application	The prototype application that is the result of this master thesis.
XES trace	A single trace with its XES events in the XES log.

Appendix B

MXML Meta Model

This appendix shows the MXML meta model from [9].

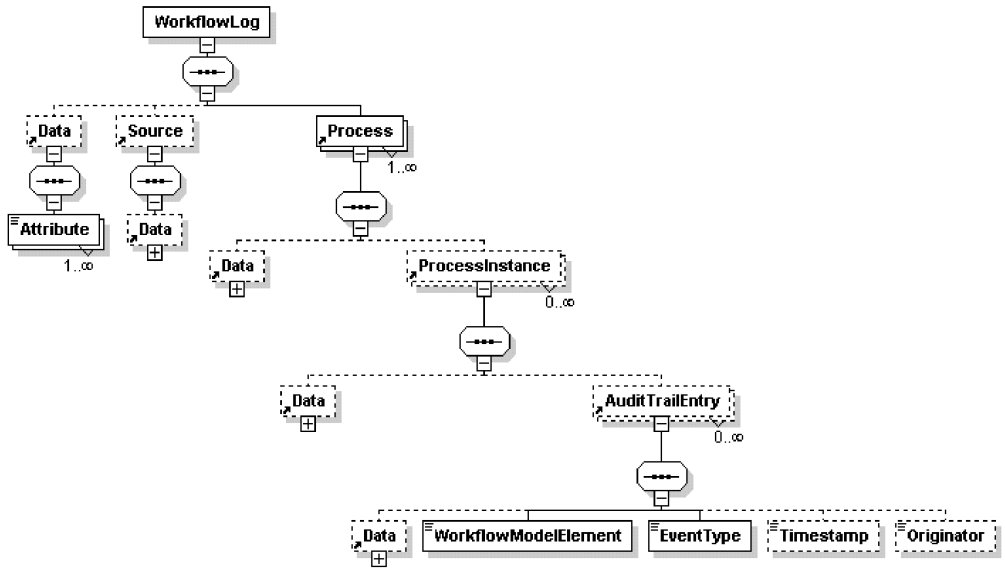


Figure B.1: MXML meta model

Appendix C

XES Schema Definitions

This appendix lists the schema definitions of the XES event log standard.

C.1 XES Schema Definition

Listing C.1 shows the XSD schema definition of the XES event log file version 1.0 revision 3. The most current definition can be found at `http://code.fluxicon.com/xes/xes.xsd`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
    qualified"
  targetNamespace="http://code.fluxicon.com/xes" xmlns:xes="http://code.fluxicon.
    com/xes">
  <!-- This file describes the XML serialization of the XES format for event log
    data. -->
  <!-- For more information about XES, visit http://code.deckfour.org/xes/ -->

  <!-- (c) 2009 by Christian W. Guenther (christian@fluxicon.com) -->

  <!-- Every XES XML Serialization needs to contain exactly one log element -->
  <xs:element name="log" type="xes:LogType"/>

  <!-- String attribute -->
  <xs:complexType name="AttributeStringType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">

      <xs:element name="string" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeStringType"/>
      <xs:element name="date" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeDateType"/>
      <xs:element name="int" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeIntType"/>
      <xs:element name="float" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeFloatType"/>
      <xs:element name="boolean" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeBooleanType"/>
    </xs:choice>
    <xs:attribute name="key" use="required" type="xs:Name"/>
    <xs:attribute name="value" use="required" type="xs:string"/>
  </xs:complexType>


  <!-- Date attribute -->
  <xs:complexType name="AttributeDateType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="string" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeStringType"/>
```

```
    <xs:element name="date" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeDateType"/>
    <xs:element name="int" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeIntType"/>
    <xs:element name="float" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeFloatType"/>
    <xs:element name="boolean" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeBooleanType"/>
  </xs:choice>

  <xs:attribute name="key" use="required" type="xs:Name"/>
  <xs:attribute name="value" use="required" type="xs:dateTime"/>
</xs:complexType>

<!-- Integer attribute -->
<xs:complexType name="AttributeIntType">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="string" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeStringType"/>
    <xs:element name="date" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeDateType"/>
    <xs:element name="int" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeIntType"/>

    <xs:element name="float" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeFloatType"/>
    <xs:element name="boolean" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeBooleanType"/>
  </xs:choice>
  <xs:attribute name="key" use="required" type="xs:Name"/>
  <xs:attribute name="value" use="required" type="xs:long"/>
</xs:complexType>

<!-- Floating-point attribute -->
<xs:complexType name="AttributeFloatType">
  <xs:choice minOccurs="0" maxOccurs="unbounded">

    <xs:element name="string" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeStringType"/>
    <xs:element name="date" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeDateType"/>
    <xs:element name="int" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeIntType"/>
    <xs:element name="float" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeFloatType"/>
    <xs:element name="boolean" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeBooleanType"/>
  </xs:choice>
  <xs:attribute name="key" use="required" type="xs:Name"/>
  <xs:attribute name="value" use="required" type="xs:double"/>
</xs:complexType>


<!-- Boolean attribute -->
<xs:complexType name="AttributeBooleanType">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="string" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeStringType"/>
    <xs:element name="date" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeDateType"/>
    <xs:element name="int" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeIntType"/>
    <xs:element name="float" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeFloatType"/>
    <xs:element name="boolean" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeBooleanType"/>
  </xs:choice>
```

```
  <xs:attribute name="key" use="required" type="xs:Name"/>
  <xs:attribute name="value" use="required" type="xs:boolean"/>
</xs:complexType>

<!-- Extension definition -->
<xs:complexType name="ExtensionType">
  <xs:attribute name="name" use="required" type="xs:NCName"/>
  <xs:attribute name="prefix" use="required" type="xs:NCName"/>
  <xs:attribute name="uri" use="required" type="xs:anyURI"/>
</xs:complexType>


<!-- Globals definition -->
<xs:complexType name="GlobalsType">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="string" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeStringType"/>
    <xs:element name="date" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeDateType"/>
    <xs:element name="int" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeIntType"/>
    <xs:element name="float" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeFloatType"/>
    <xs:element name="boolean" minOccurs="0" maxOccurs="unbounded" type="
        xes:AttributeBooleanType"/>
  </xs:choice>

  <xs:attribute name="scope" type="xs:NCName" use="required"/>
</xs:complexType>

<!-- Classifier definition -->
<xs:complexType name="ClassifierType">
  <xs:attribute name="name" type="xs:NCName" use="required"/>
  <xs:attribute name="keys" type="xs:token" use="required"/>
</xs:complexType>

<!-- Logs may contain attributes and traces -->

<xs:complexType name="LogType">
  <xs:sequence>
    <xs:element name="extension" minOccurs="0" maxOccurs="unbounded" type="
        xes:ExtensionType"/>
    <xs:element name="global" minOccurs="0" maxOccurs="2" type="xes:GlobalsType"/
        >
    <xs:element name="classifier" minOccurs="0" maxOccurs="unbounded"/>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="string" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeStringType"/>
      <xs:element name="date" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeDateType"/>
      <xs:element name="int" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeIntType"/>

      <xs:element name="float" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeFloatType"/>
      <xs:element name="boolean" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeBooleanType"/>
    </xs:choice>
    <xs:element name="trace" maxOccurs="unbounded" type="xes:TraceType"/>
  </xs:sequence>
  <xs:attribute name="xes.version" type="xs:decimal" use="required"/>
  <xs:attribute name="xes.features" type="xs:token" use="required"/>
  <xs:attribute name="openxes.version" type="xs:string" use="required"/>
</xs:complexType>
```

```
  <!-- Traces may contain attributes and events -->
  <xs:complexType name="TraceType">
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="string" minOccurs="0" maxOccurs="unbounded" type="
            xes:AttributeStringType"/>
        <xs:element name="date" minOccurs="0" maxOccurs="unbounded" type="
            xes:AttributeDateType"/>
        <xs:element name="int" minOccurs="0" maxOccurs="unbounded" type="
            xes:AttributeIntType"/>
        <xs:element name="float" minOccurs="0" maxOccurs="unbounded" type="
            xes:AttributeFloatType"/>
        <xs:element name="boolean" minOccurs="0" maxOccurs="unbounded" type="
            xes:AttributeBooleanType"/>

      </xs:choice>
      <xs:element name="event" maxOccurs="unbounded" type="xes:EventType"/>
    </xs:sequence>
  </xs:complexType>

  <!-- Events may contain attributes -->
  <xs:complexType name="EventType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="string" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeStringType"/>
      <xs:element name="date" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeDateType"/>

      <xs:element name="int" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeIntType"/>
      <xs:element name="float" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeFloatType"/>
      <xs:element name="boolean" minOccurs="0" maxOccurs="unbounded" type="
          xes:AttributeBooleanType"/>
    </xs:choice>
  </xs:complexType>

</xs:schema>
```

Listing C.1: XES event log XSD Schema

C.2 XES Extension Schema Definition

Listing C.2 shows the XSD extension schema definition of the XES event log file version 1.0 revision 3. The most current definition can be found at `http://code.fluxicon.com/xes/xes.xsd`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
    qualified">
  <!-- This file describes the  serialization for extensions of the XES format for
      event log data. -->
  <!-- For more information about XES, visit http://code.fluxicon.com/xes/ -->

  <!-- (c) 2009 by Christian W. Guenther (christian@fluxicon.com) -->

  <!-- Any extension definition has an xesextension root element. -->
  <!-- Child elements are containers, which define attributes for -->
  <!-- the log, trace, event, and meta level of the XES -->
  <!-- type hierarchy. -->
  <!-- All of these containers are optional. -->

  <!-- The root element further has attributes, defining: -->
  <!--  * The name of the extension. -->
  <!--  * A unique prefix string for  attributes defined by this -->
  <!--    extension. -->
```

```
<!--  * A unique URI of this extension, holding the XESEXT -->
<!--    definition file. -->
<xs:element name="xesextension">
  <xs:complexType>
    <xs:sequence>

      <xs:element minOccurs="0" maxOccurs="1" ref="log"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="trace"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="event"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="meta"/>
    </xs:sequence>
    <xs:attribute name="name" use="required" type="xs:NCName"/>
    <xs:attribute name="prefix" use="required" type="xs:NCName"/>
    <xs:attribute name="uri" use="required" type="xs:anyURI"/>
  </xs:complexType>

</xs:element>

<!-- Container tag for the definition of log attributes. -->
<xs:element name="log">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="string" type="AttributeType"/>
      <xs:element name="date" type="AttributeType"/>
      <xs:element name="int" type="AttributeType"/>
      <xs:element name="float" type="AttributeType"/>

      <xs:element name="boolean" type="AttributeType"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<!-- Container tag for the definition of trace attributes. -->
<xs:element name="trace">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="string" type="AttributeType"/>

      <xs:element name="date" type="AttributeType"/>
      <xs:element name="int" type="AttributeType"/>
      <xs:element name="float" type="AttributeType"/>
      <xs:element name="boolean" type="AttributeType"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<!-- Container tag for the definition of event attributes. -->
<xs:element name="event">

  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="string" type="AttributeType"/>
      <xs:element name="date" type="AttributeType"/>
      <xs:element name="int" type="AttributeType"/>
      <xs:element name="float" type="AttributeType"/>
      <xs:element name="boolean" type="AttributeType"/>
    </xs:choice>
  </xs:complexType>

</xs:element>

<!-- Container tag for the definition of meta attributes. -->
<xs:element name="meta">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="string" type="AttributeType"/>
      <xs:element name="date" type="AttributeType"/>
```

```
        <xs:element name="int" type="AttributeType"/>
        <xs:element name="float" type="AttributeType"/>

        <xs:element name="boolean" type="AttributeType"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <!-- Attribute -->
  <xs:complexType name="AttributeType">
    <xs:sequence>
      <xs:element name="alias" type="AliasType" minOccurs="0" maxOccurs="unbounded"
          />
    </xs:sequence>

    <xs:attribute name="key" use="required" type="xs:Name"/>
  </xs:complexType>

  <!--  Alias definition, defining a mapping alias for an attribute -->
  <xs:complexType name="AliasType">
    <xs:attribute name="mapping" use="required" type="xs:NCName"/>
    <xs:attribute name="name" use="required" type="xs:string"/>
  </xs:complexType>

</xs:schema>
```

Listing C.2: XES extension XSD Schema

Appendix D

Case Study Details

This appendix shows more details about the mapping definition for the case studies performed in Chapter 6.

D.1 Case Study 1

This section contains more details on case study 1: data exported from an SAP system.

D.1.1 Conversion Definition

The detailed conversion definition for case study 1 is shown in Figures D.1-D.11.

D.1.2 Performance Data

This section shows the raw performance data used to create the graphs of Section 6.1.3. Table D.1 shows the characteristics for each conversion execution. Table D.2 provides detailed information of the duration of each step for a selection of executions.

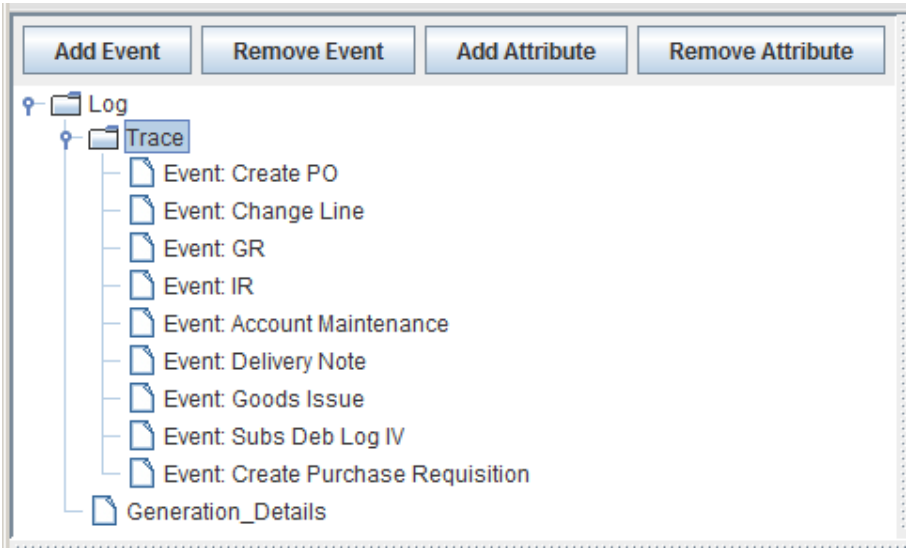
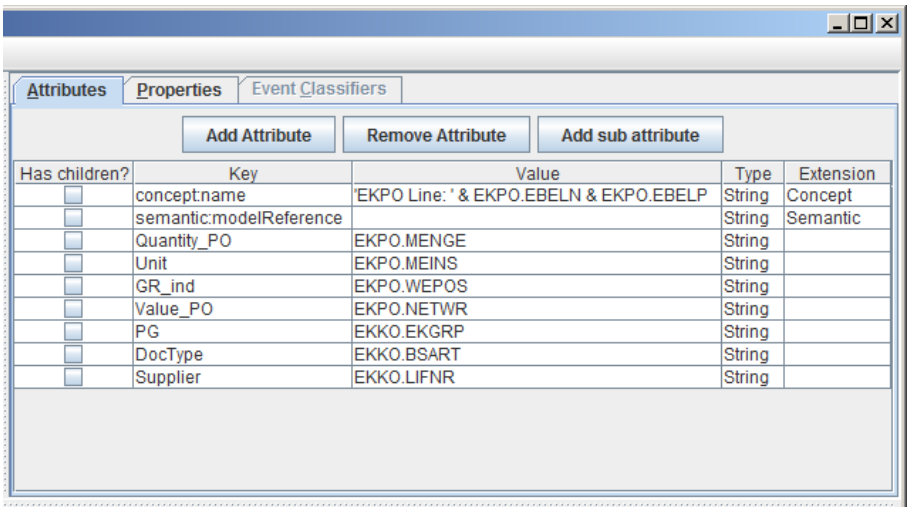
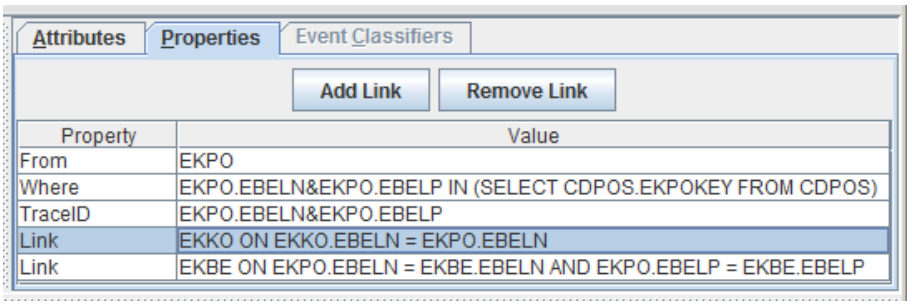


Figure D.1: The mapping tree with all the defined events for case study 1.

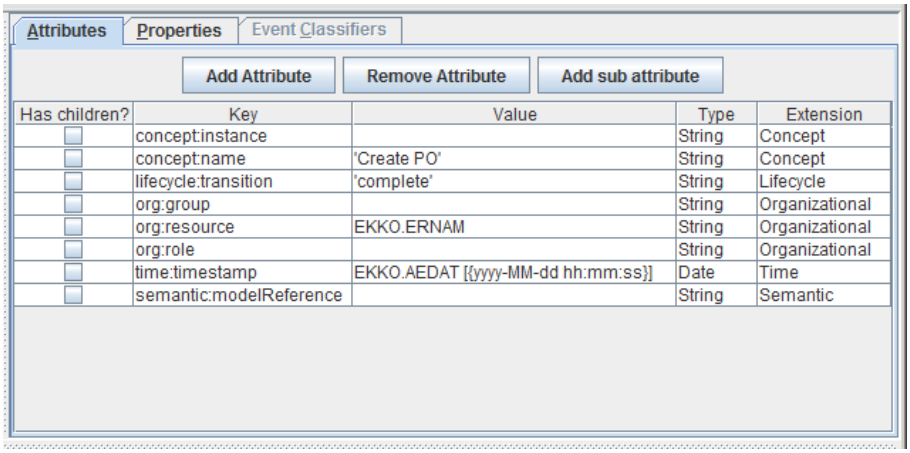


(a) Trace attributes specification

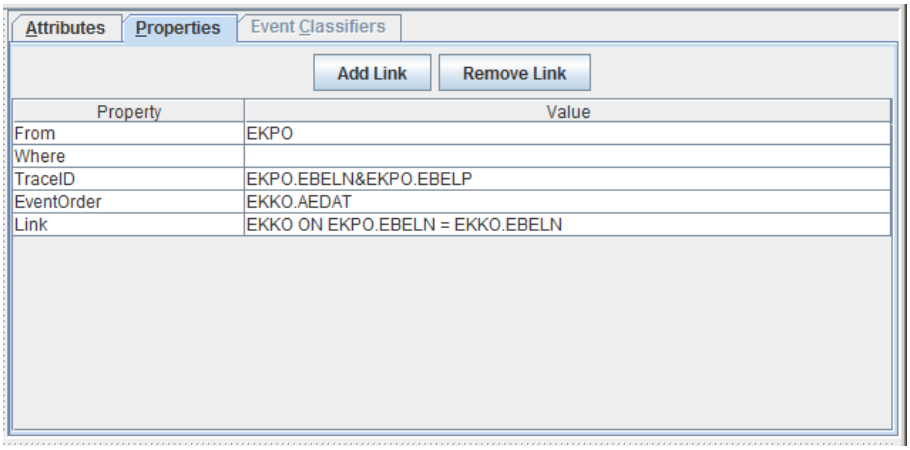


(b) Trace properties specification

Figure D.2: Trace element conversion definition for case study 1.



(a) Create purchase order event attributes specification

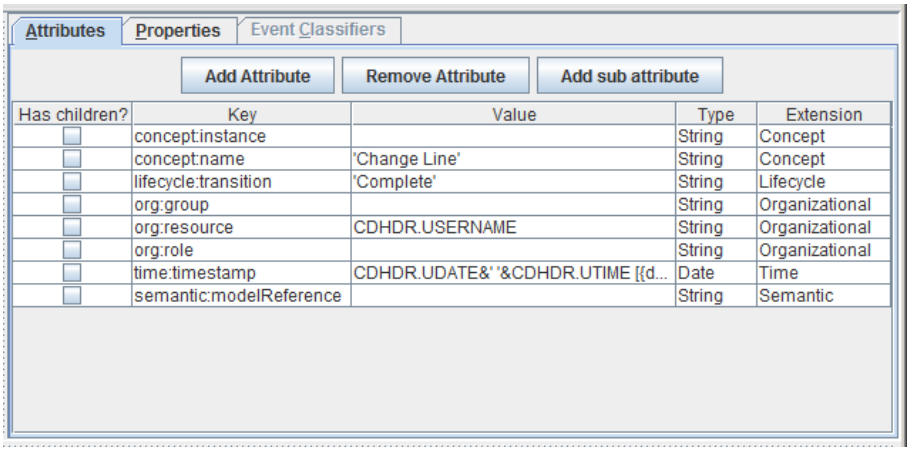


(b) Create purchase order event properties specification

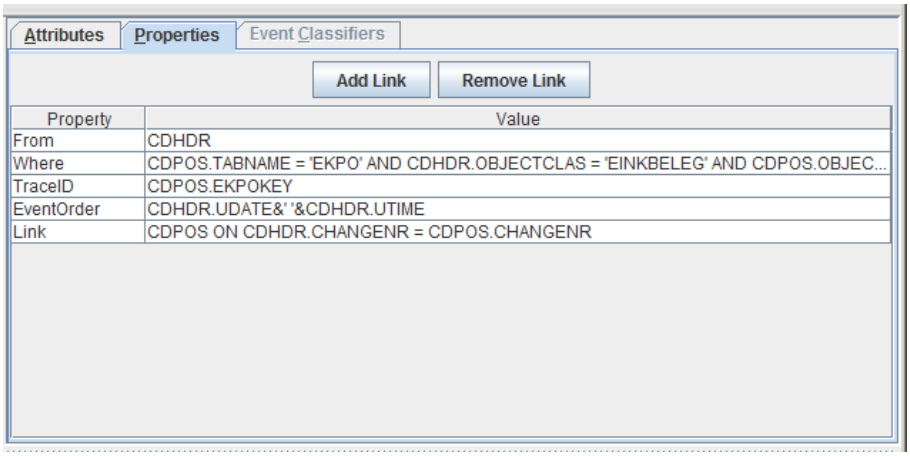
Figure D.3: Create purchase order event conversion definition for case study 1.

Table D.1: Overall duration of the execution of different runs.

Traces	Events	Events/Trace	Total Duration	Seconds	Sec./Trace	Sec./Event
1	20	20,00	0:01:20	80,00	80,00	4,00
10	283	28,30	0:02:06	126,00	12,60	0,45
100	1624	16,24	0:08:33	513,00	5,13	0,32
500	7972	15,94	0:30:29	1829,00	3,66	0,23
500	7972	15,94	0:13:01	781,00	1,56	0,10
500	7972	15,94	0:13:07	787,00	1,57	0,10
1000	14336	14,34	0:23:38	1418,00	1,42	0,10
1000	14336	14,34	0:23:40	1420,00	1,42	0,10
1000	14336	14,34	0:23:34	1414,00	1,41	0,10
2000	28520	14,26	0:45:45	2745,00	1,37	0,10
4000	54016	13,50	1:28:11	5291,00	1,32	0,10

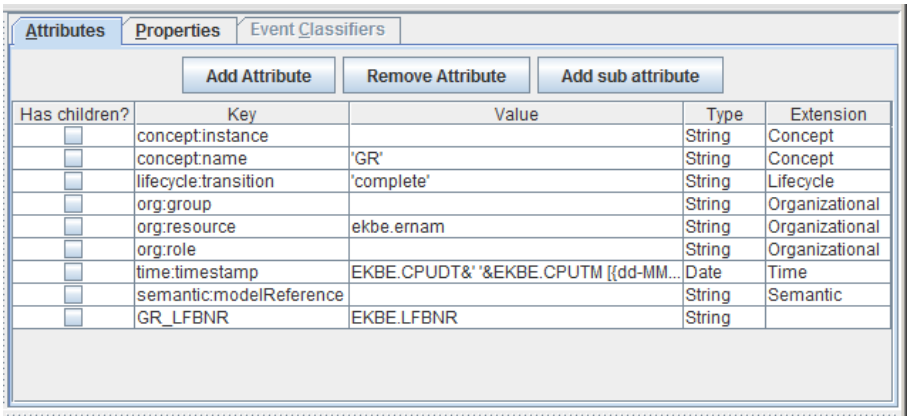


(a) Change line event attributes specification

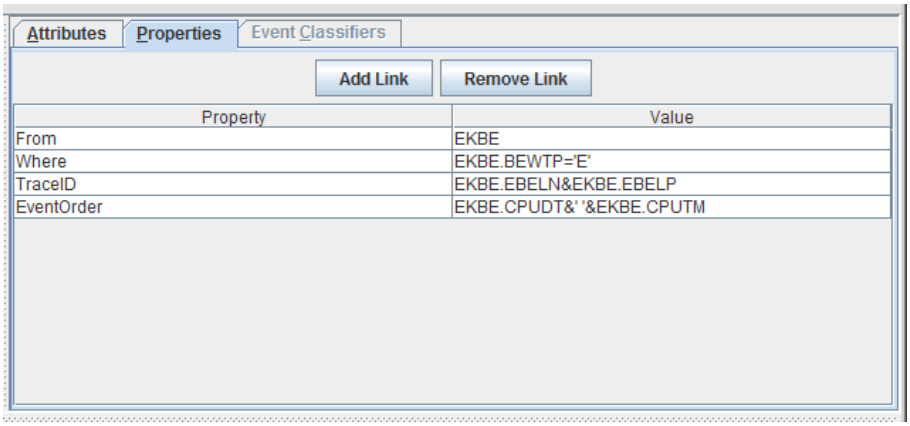


(b) Change line event properties specification

Figure D.4: Change line event conversion definition for case study 1.

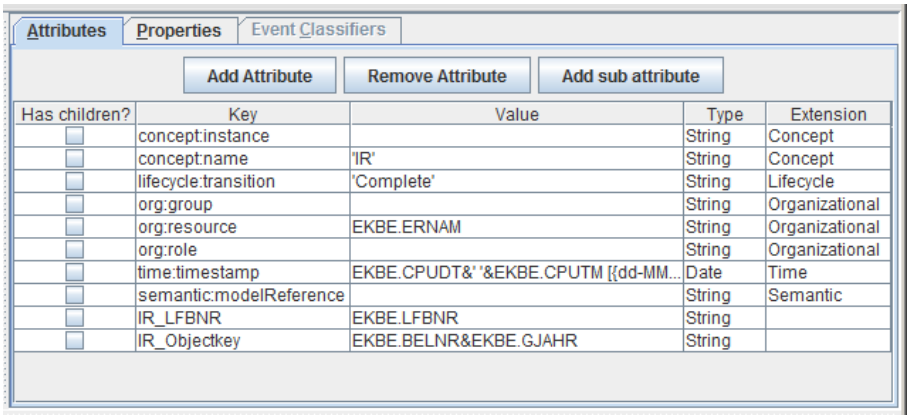


(a) Goods receipt event attributes specification



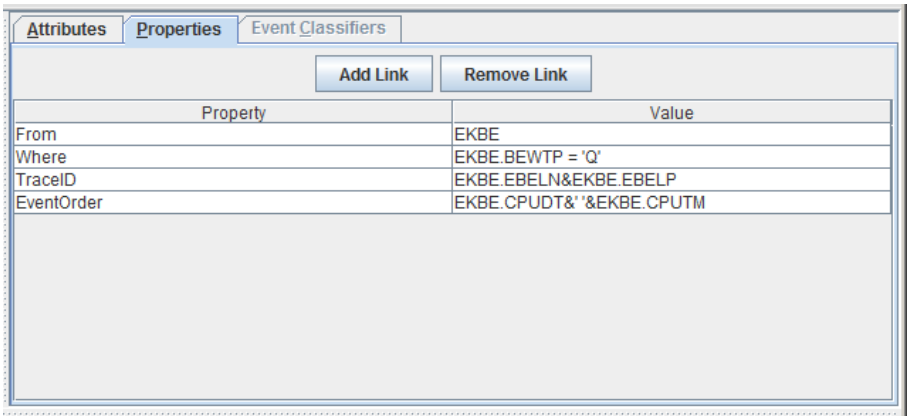
(b) Goods receipt event properties specification

Figure D.5: Goods receipt event conversion definition for case study 1.



Has children?	Key	Value	Type	Extension
☐	concept:instance		String	Concept
☐	concept:name	'IR'	String	Concept
☐	lifecycle:transition	'Complete'	String	Lifecycle
☐	org:group		String	Organizational
☐	org:resource	EKBE.ERNAM	String	Organizational
☐	org:role		String	Organizational
☐	time:timestamp	EKBE.CPUDT&' '&EKBE.CPUTM [{dd-MM...	Date	Time
☐	semantic:modelReference		String	Semantic
☐	IR_LFBNR	EKBE.LFBNR	String	
☐	IR_Objectkey	EKBE.BELNR&EKBE.GJAHR	String	

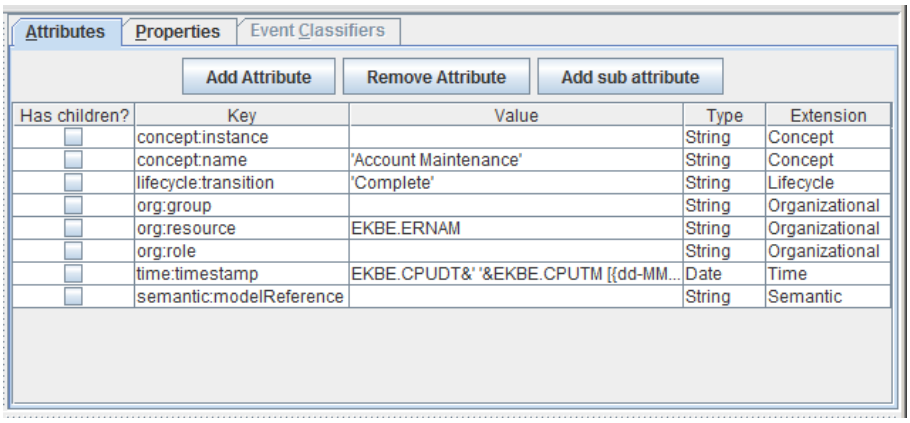
(a) Invoice receipt event attributes specification



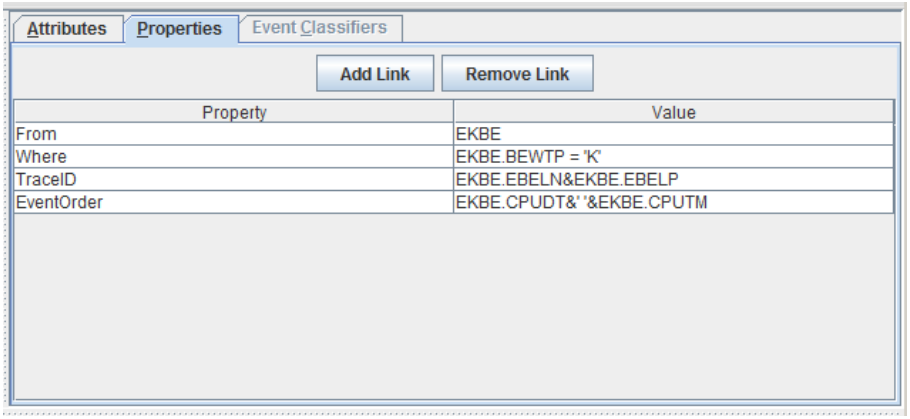
Property	Value
From	EKBE
Where	EKBE.BEWTP = 'Q'
TraceID	EKBE.EBELN&EKBE.EBELP
EventOrder	EKBE.CPUDT&' '&EKBE.CPUTM

(b) Invoice receipt event properties specification

Figure D.6: Invoice receipt event conversion definition for case study 1.

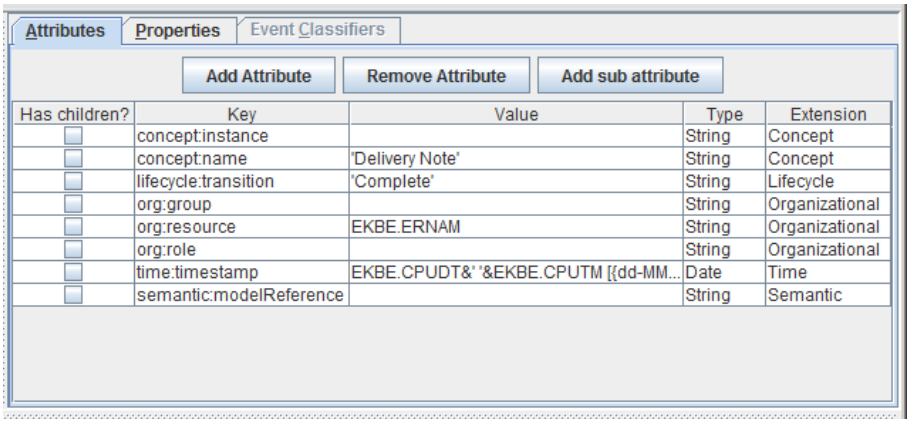


(a) Account maintenance event attributes specification

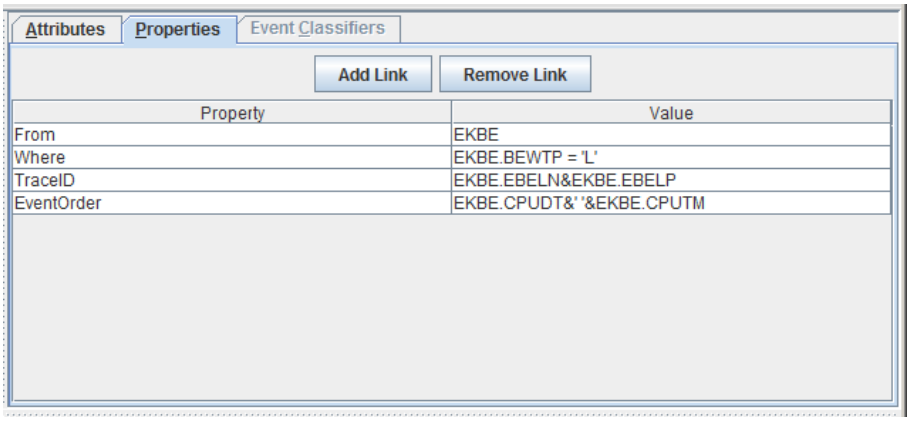


(b) Account maintenance event properties specification

Figure D.7: Account maintenance event conversion definition for case study 1.

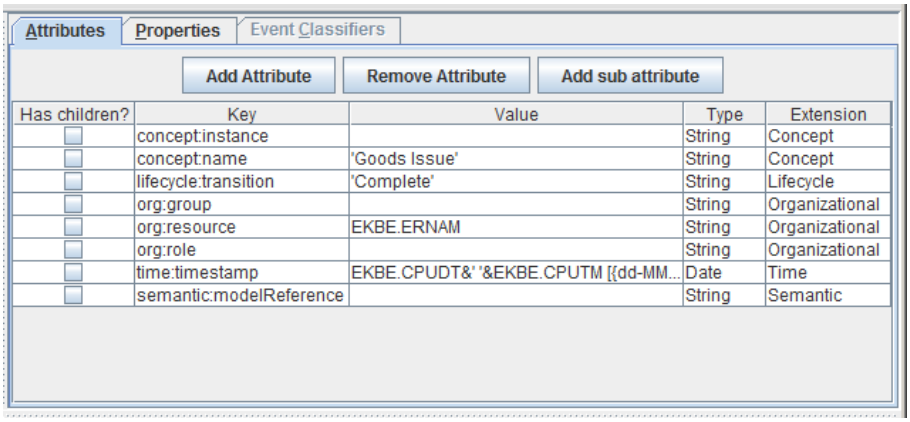


(a) Delivery note event attributes specification

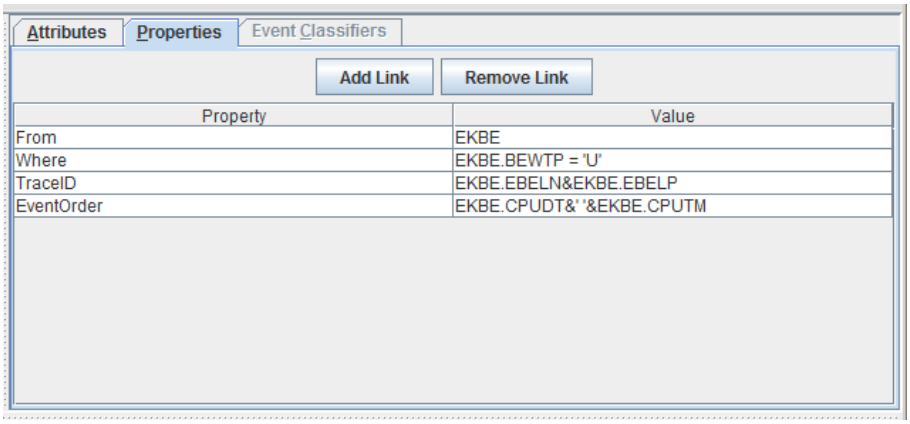


(b) Delivery note event properties specification

Figure D.8: Delivery note event conversion definition for case study 1.

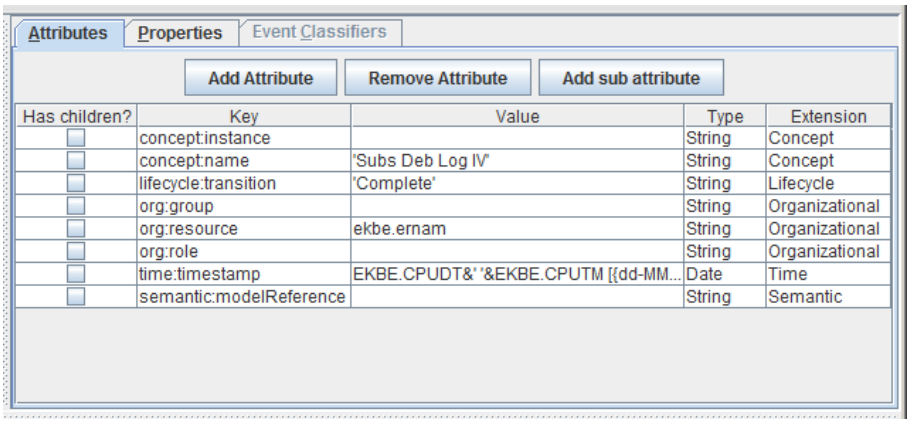


(a) Goods issue event attributes specification

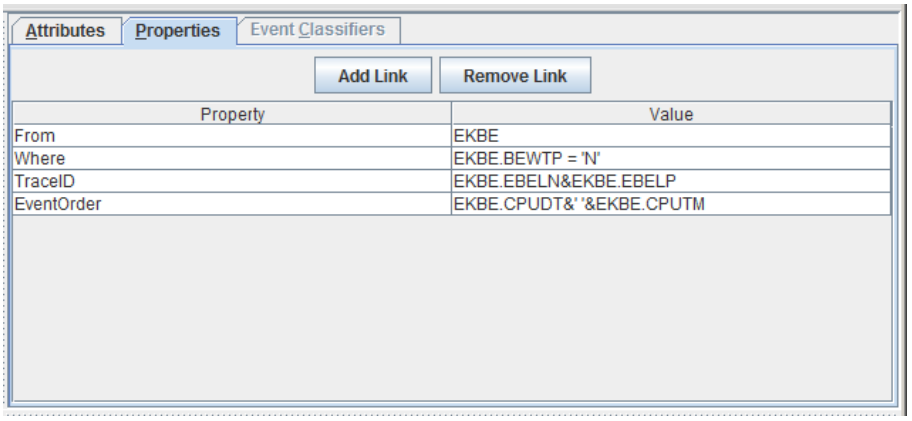


(b) Goods issue event properties specification

Figure D.9: Goods issue event conversion definition for case study 1.

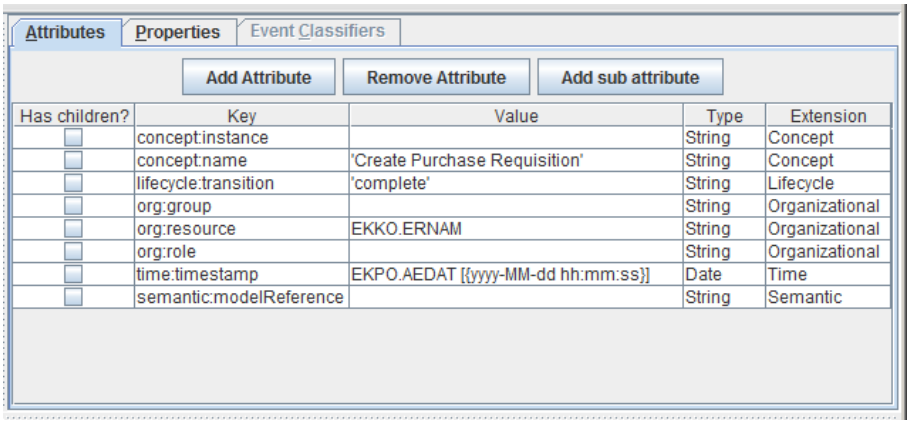


(a) Subs Deb Log IV event attributes specification

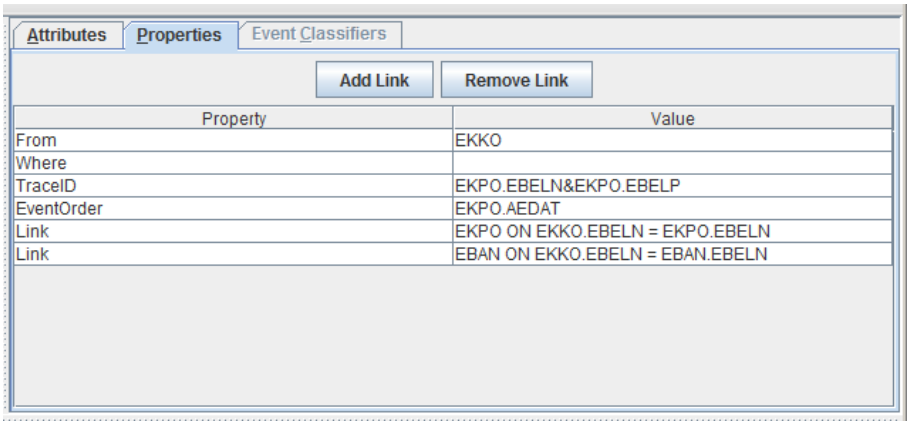


(b) Subs Deb Log IV event properties specification

Figure D.10: Subs Deb Log IV event conversion definition for case study 1.



(a) Create purchase requisition event attributes specification



(b) Create purchase requisition event properties specification

Figure D.11: Create purchase requisition event conversion definition for case study 1.

Table D.2: Duration of the execution of selected runs split by phase.

Step	**Description**	**500 traces**	**500 traces II**	**500 traces III**	**1000 traces**	**1000 traces II**	**1000 traces III**	**2000 traces**	**4000 traces**
1	Initialization	0:00:01	0:00:01	0:00:05	0:00:03	0:00:02	0:00:01	0:00:01	0:00:01
2	Log	0:00:00	0:00:01	0:00:01	0:00:00	0:00:00	0:00:01	0:00:00	0:00:00
3	Trace	0:01:53	0:00:45	0:00:45	0:01:04	0:01:05	0:01:03	0:01:45	0:03:07
4	Event: Create PO	0:03:44	0:03:05	0:03:05	0:06:02	0:06:04	0:06:04	0:11:58	0:23:34
5	Event: Change Line	0:06:13	0:01:43	0:01:44	0:02:56	0:02:55	0:02:54	0:06:13	0:10:49
6	Event: GR	0:03:42	0:01:42	0:01:42	0:03:02	0:03:02	0:03:01	0:05:39	0:10:58
7	Event: IR	0:03:52	0:01:37	0:01:37	0:03:02	0:03:02	0:03:01	0:05:42	0:10:44
8	Event: Account Maintenance	0:00:39	0:00:13	0:00:13	0:00:25	0:00:25	0:00:26	0:00:34	0:00:46
9	Event: Delivery Note	0:00:03	0:00:04	0:00:04	0:00:04	0:00:04	0:00:03	0:00:04	0:00:04
10	Event: Goods Issue	0:00:07	0:00:07	0:00:07	0:00:12	0:00:12	0:00:12	0:00:21	0:00:47
11	Event: Subs Deb Log IV	0:00:05	0:00:05	0:00:04	0:00:05	0:00:05	0:00:05	0:00:07	0:00:09
12	Event: Create Purchase Requisition	0:03:09	0:02:59	0:02:59	0:05:48	0:05:51	0:05:49	0:11:33	0:23:44
13	Converting Cache DB to XES	0:07:01	0:00:40	0:00:41	0:00:55	0:00:54	0:00:54	0:01:48	0:03:28

D.2 Case Study 2

This section contains more details on case study 2: data exported from a custom workflow system.

D.2.1 Conversion Definition

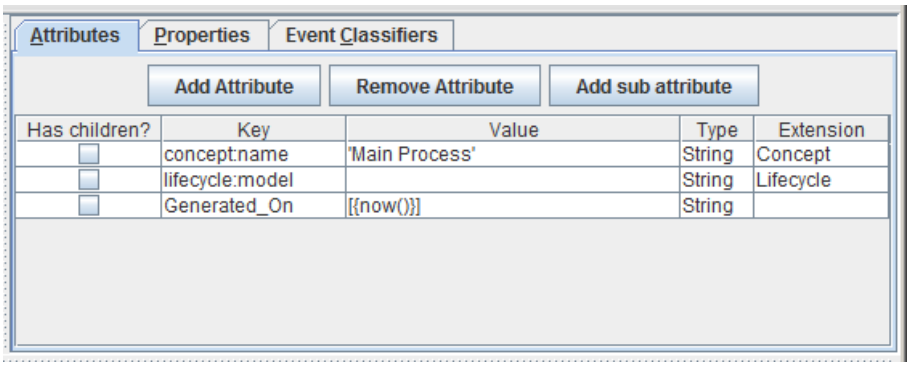
The detailed conversion definition for case study 2 is shown in Figures D.12-D.15.

D.2.2 Performance Data

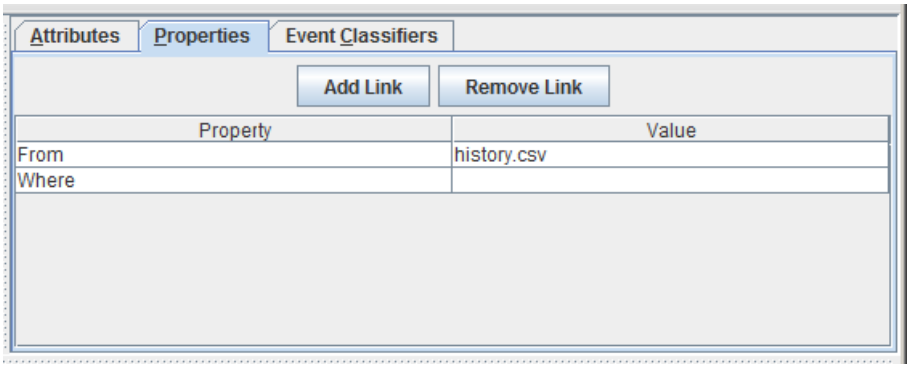
This section shows the raw performance data used to create the graphs of Section 6.2.3. Table D.3 shows the characteristics for each conversion execution. Table D.4 provides detailed information of the duration of each step for a selection of executions.

Table D.3: Duration of the execution of selected runs.

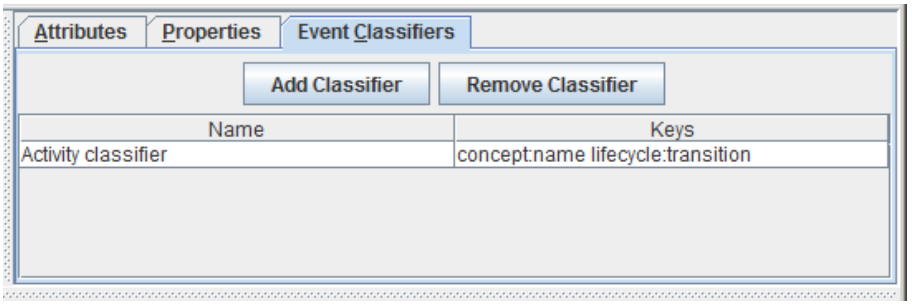
Traces	Events	events/trace	Total Duration	Seconds	sec/trace	sec/event
10	28	2,80	0:26:33	1593	159,30	56,89
20	94	4,70	0:30:39	1839	91,95	19,56
50	1188	23,76	0:43:13	2593	51,86	2,18
100	2394	23,94	1:01:37	3697	36,97	1,54
200	5234	26,17	1:40:11	6011	30,06	1,15
500	13342	26,68	3:35:52	12952	25,90	0,97



(a) Log attributes specification



(b) Log properties specification

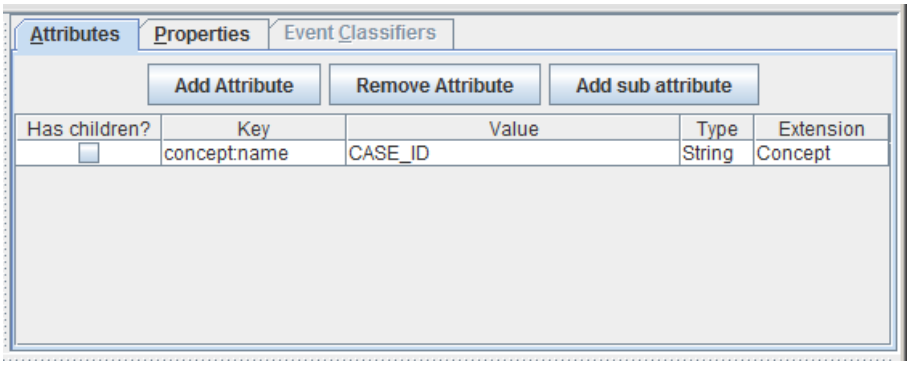


(c) Log event classifier specification

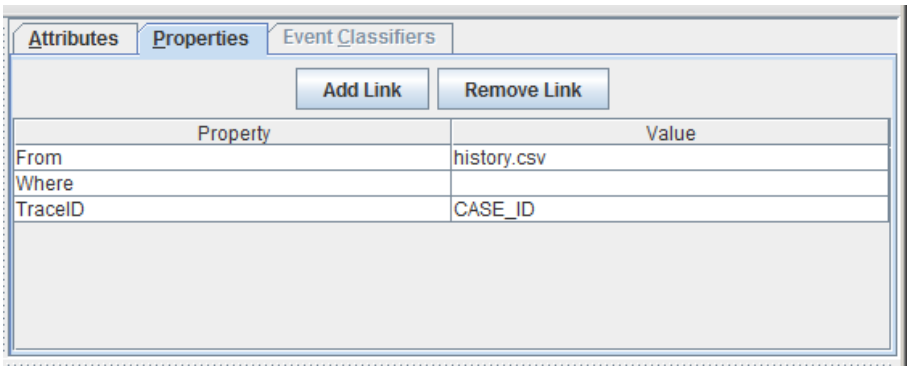
Figure D.12: Log element conversion definition for case study 2.

Table D.4: Duration of the execution of selected runs split by phase.

Step	Description	100 traces	200 traces	500 traces
1	Initialization	0:00:01	0:00:01	0:00:01
2	Log	0:02:59	0:03:00	0:03:01
3	Trace	0:05:03	0:04:59	0:05:03
4	Event: Start	0:26:41	0:45:54	1:43:21
5	Event: Complete	0:26:42	0:45:53	1:43:18
6	Converting Cache DB to XES	0:00:11	0:00:24	0:01:09

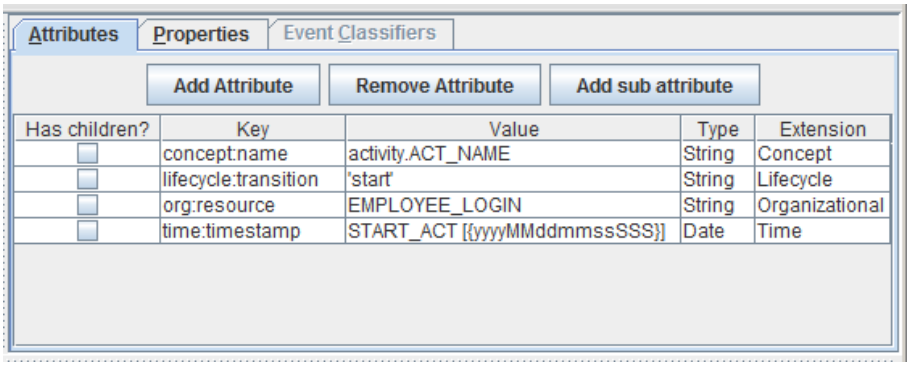


(a) Trace attributes specification

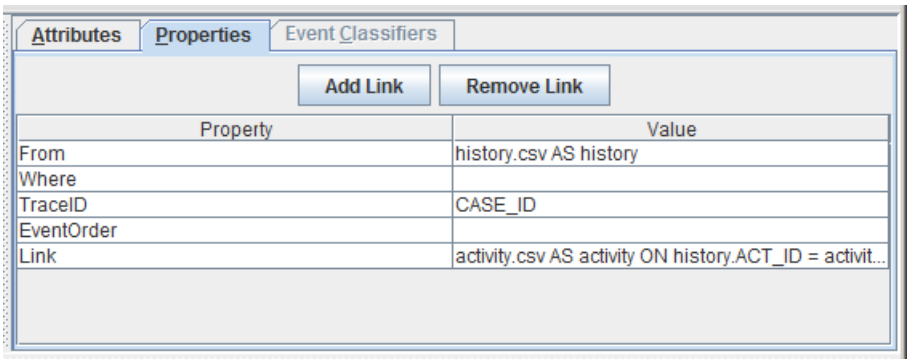


(b) Trace properties specification

Figure D.13: Trace element conversion definition for case study 2.

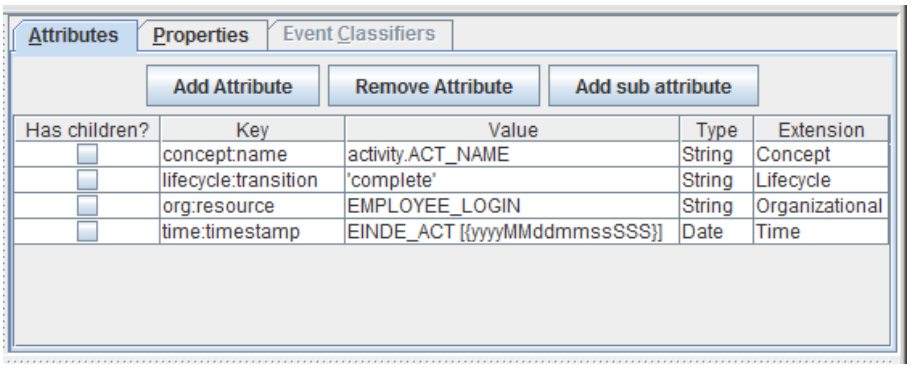


(a) Start event attributes specification

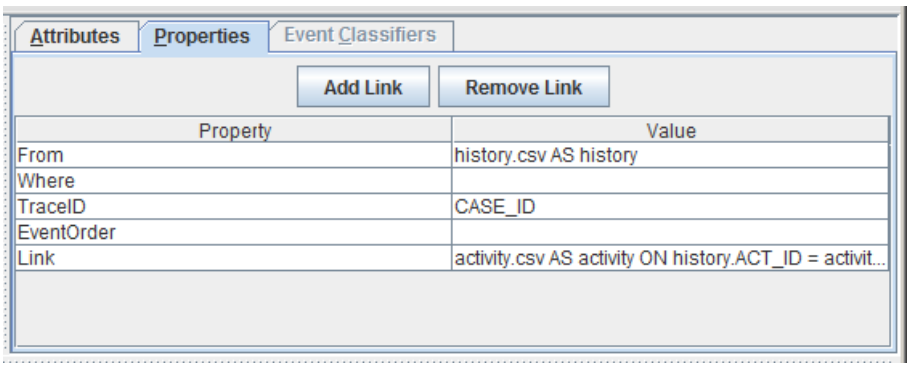


(b) Start event properties specification

Figure D.14: Start event conversion definition for case study 2.



(a) Complete event attributes specification



(b) Complete event properties specification

Figure D.15: Complete event conversion definition for case study 2.