University of Tartu
Faculty of Mathematics and Computer Science
Institute of Computer Science

Tanel Teinemaa

Discovering Entities in Process Execution Logs

Bachelor's thesis (6 ECTS)

Supervisor: Viara Popova
Co-supervisor: Meelis Kull

Author: ..... "....." May 2012
Supervisor: ..... "....." May 2012
Co-supervisor: ..... "....." May 2012
Allowed to defend
Professor: ..... "....." May 2012

Tartu 2012

TABLE OF CONTENTS

INTRODUCTION

The topic of this thesis is Discovering Entities in Process Execution Logs. The thesis is written in the frames of the Artifact-Centric Service Interoperation (ACSI) project. The goal of this project is to enable faster, smoother Internet services to be provided by developing new techniques and tools.

Process mining techniques are used to analyze the new systems and tools on which the project focuses. Computer scientists have developed various methods to process large amounts of data in order to extract useful knowledge from it. These methods are called data mining. Process mining is a branch of data mining and is based on extracting meaningful knowledge from the event logs.

The contribution of this thesis is to develop and implement a new method for discovering entities and candidate artifacts in the event logs. An artifact describes a class of similar objects [7], e.g., all purchase orders in a particular shop. This method would make it possible to use the existing process mining methods, such as life cycle discovery (to build an artifact model), conformance checking (against an existing model) etc., for the analysis of the artifact-centric models. In addition, this would allow the development of new types of analysis such as conformance checking of the artifact structure. For example, one of the tools developed in relation to the ACSI project is the conformance checker, which will be responsible for checking the conformance of Internet services against the artifact-centric business process models. Discovering entities in process execution logs (event logs) is a part of this conformance checker.

The new method should perform the following tasks. Firstly, the method should take raw logs in the format of XES as input and extract event type tables from the logs. Secondly, the method should find the candidate keys for these tables. After that, the user is asked to choose a primary key from the set of candidate keys for each table. After the primary keys have been selected, the method should group together the event types that share the same primary keys and integrate them into one entity. Finally, the method should show the formed entities to the user as output.

The set of entities formed by the new method will be a set of candidate artifacts. Other methods can be used to decide which are the most important ones or input from the user can be given. In practice it is partly subjective which entities are considered important enough to become artifacts. Heuristics can be used which require additional analysis, such as discovering relationships between the entities. Also, input from the user can be asked.

After the main entities (called primary entities) become artifacts, the rest of the entities (called secondary entities) can be joined with one or more of these artifacts. The process of choosing the artifacts is outside of the scope of this thesis.

The new method that is presented in this thesis is implemented as a plug-in for ProM, written in the Java programming language. ProM [14] is a generic open-source Java framework for implementing process mining algorithms as plug-ins. It also uses open source implementation of TANE algorithm [8], which will be used for finding functional dependencies and candidate keys.

The thesis is divided into four chapters. In the first chapter, the relevant terms are explained, the used implementation framework is introduced and the example of a CD-shop that is also used for testing the method is described. In the second chapter, the tasks of the new method are explained in detail. In the third chapter, the implementation of the method is described. In the last chapter, the results from testing the method are analysed.

1. BACKGROUND

1.1. Process Mining

Data mining is a collection of a wide variety of techniques for examining large preexisting databases in order to generate new information [9]. Process mining is data mining of event logs instead of databases. This means process mining is a collection of techniques that enable the analysis and improvement of business processes based on event logs.

Event logs store some information about the execution of processes. A process is a series of actions taken in order to achieve a particular result. The workflow is the sequence of processes through which a piece of work passes from initiation to completion [9]. A workflow management system is a computer system that manages and defines a series of activities to produce the final outcome. Workflow management systems can also record the information about the execution of the activities into event logs.

Some reasons why the process mining is useful to businesses are new legislation that is forcing organizations to follow their business activities more closely and the constant pressure to improve the performance and efficiency of business processes [1].

Process mining can be used for reverse engineering the business model from logs and to analyze and improve the overall workflow and remove the bottlenecks. Process mining can also be used for social networking discovery to find the connection and interaction patterns between people, to assign potential roles to them and to divide these people into groups. Process mining can also be used for conformance checking and auditing the already existing business model against the event logs. The purpose of conformance checking is to ascertain the conformance of the execution log of an existing system with respect to a given model and to pinpoint deviations between the logs and the model [15].

1.2. CD-Shop Example

In this section, we are going to introduce an example, which will later be used throughout the thesis as a running example. Let us describe a common process in an online CD-shop, when the customer buys music from it. At first, the customer places a purchase order and it gets either approved or rejected. If approved, the required CDs are ordered from the suppliers (material orders) and in the end, these items are delivered to the customer. CDs from purchase orders can be split between multiple material orders. If

supplier cannot successfully complete the material order then material orders can either be reordered or canceled.

This CD-shop example [2] is also used in the ACSI project. Additionally, there exists a set of artificially generated process execution logs based on this online CD-shop example. These logs are suitable for testing the new method. More information about generating the logs can be found in section 4.2.

1.3. Event Logs

As mentioned above, event logs [4] store some information about the execution of processes. An event log records events from a certain process or processes. An event reflects an execution of an activity associated with the process. A process instance, i.e. a case, describes one execution of a process. For example let us have a process as shown informally on figure 1.1. A specific process instance can for example describe the following: Customer John Smith orders 2 CDs, but as he does not pay for them within the required period of time, the order is canceled.

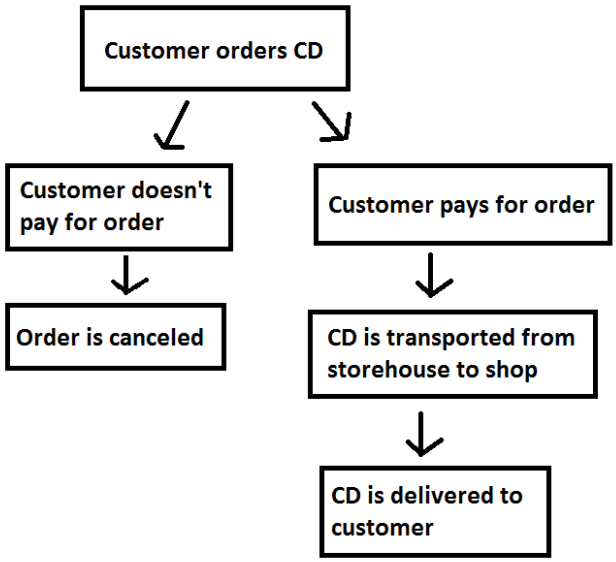

Figure 1.1. Example of a process in the CD-shop.

The events of a process instance are grouped as a single conceptual entity called a trace [15]. An example of a trace containing four events can be the following: 1) Customer orders a CD. 2) Customer pays for the order. 3) The CD is transported from the storehouse to the shop. 4) The CD is delivered to the customer. This order of events is usually defined

by the timestamp attribute (if timestamp is present for these events). The timestamp contains the occurring date and time for a single event.

There are some common attributes for events. One of them is the timestamp and there are also event type and resource. The event type determines the activity and its state (e.g. “creating new request completed” or “done saving file”). Resource is simply the executor of the process: either the user of the system or the system itself.

Traditional process mining is process-centric and it assumes that the event logs belong to a single process and are organized in cases where each case describes one execution of the process. For artifact-centric process mining, it is not required for event logs to be organized into cases. If the event logs are not organized into cases and are in one trace, then we call these logs raw logs.

There are two log formats for ProM: the XES and its ancestor MXML. These log formats were created to standardize the storing of the information in an event log. Both XES and MXML are in XML notation.

On figure 1.2 there is a part of a XES file as an example. The traces in this example are between the <trace> tags. A trace contains several events from the same process instance. The events are between the <event> tags. The string element which belongs to the event element where the key=“concept:name” stands for the event type attribute. The value of this element determines the type of this event. In the online CD-shop event logs, the purchase orders are named quotes and material orders are simply named orders.

```
...
<trace>
     <string key="concept:name" value="481"/>
     <event>
          <int key="Id" value="481"/>
          <string key="org:resource" value="Ann"/>
          <date key="time:timestamp"
               value="2010-10-18T15:18:29.000+02:00"/>
          <string key="concept:name"
               value="Generate request"/>
     </event>
     <event>
          <int key="Id" value="481"/>
          <string key="org:resource" value="CD shop"/>
          <date key="time:timestamp"
               value="2010-10-18T15:18:29.000+02:00"/>
          <int key="price_amount" value="20"/>
          <string key="concept:name" value="Send quote"/>
     </event>
     <event>
          <int key="Id" value="481"/>
          <string key="org:resource" value="Ann"/>
          <date key="time:timestamp"
               value="2010-10-19T00:00:14.000+02:00"/>
          <string key="concept:name" value="Reject quote"/>
     </event>
</trace>
<trace>
     <string key="concept:name" value="482"/>
     <event>
          <int key="Id" value="482"/>
          <string key="org:resource" value="Frank"/>
          <date key="time:timestamp"
               value="2010-10-18T16:41:49.000+02:00"/>
          <string key="concept:name"
               value="Generate request"/>
     </event>
...
```

Figure 1.2. A part of a log file in XES format.

1.4. Artifact-Centric Modeling

A business process model is a representation of enterprise processes. Process models are commonly used to analyze and improve the processes.

A traditional process-centric process model is based on the workflow model and focuses on the constructs and patterns of a process model, without providing the structure or life cycle of the data related to the workflow [15].

Artifacts are business-relevant objects that are created, evolved, and (typically) archived as they pass through a business. An artifact instance is an object that participates in the process [7], for example one purchase order. Artifacts combine both data aspects and process aspects into a holistic unit, and serve as the basic building blocks from which models of business operations and processes are constructed. The artifact type includes both an information model for data about the business object during their lifetime, and a lifecycle model, describing the possible ways and timings of the tasks that can be invoked on these objects [16]. For example, the lifecycle model would include the multiple ways that the CDs can be delivered to the customer and be paid for.

Data related to the artifacts might not always be present in raw logs, but for the artifact-centric process mining we assume that at least the minimum amount of data required for the artifact-centric approach is present, such as a timestamp, event type and at least one domain-specific data attribute. These attributes form the information model for this artifact. In order to perform artifact discovery, the domain-specific attribute(s) should contain enough information to distinguish between different artifacts and instances. We call these sets of attributes as artifact identifiers. Ideally the artifact will include the artifact identifiers but it is not necessary to know which attribute(s). For example the material order might have attributes like material_order_id, process_order_id, ordered_item_id and so on, where material_order_id is the identifier. In the raw logs it might not always be trivial which attribute (or attributes) should be identifier(s) for the artifact.

Artifact-centric modeling allows multiple artifacts with independent life cycles, multiple instances of each artifact to exist in parallel and n-m relations between the artifacts. Most of the existing process mining methods cannot be used directly for artifact-centric approaches unless we know which events belong to which instance of which artifact.

For example in this artifact-centric model, the purchase order, the material order and the delivery can be the artifacts. Figure 1.3 shows informally one example of an execution of such an artifact model with two purchase orders, two material orders and two deliveries. In the artifact-centric model, items from different purchase orders may be combined into one material order as well as one purchase order may require multiple material orders to be made.

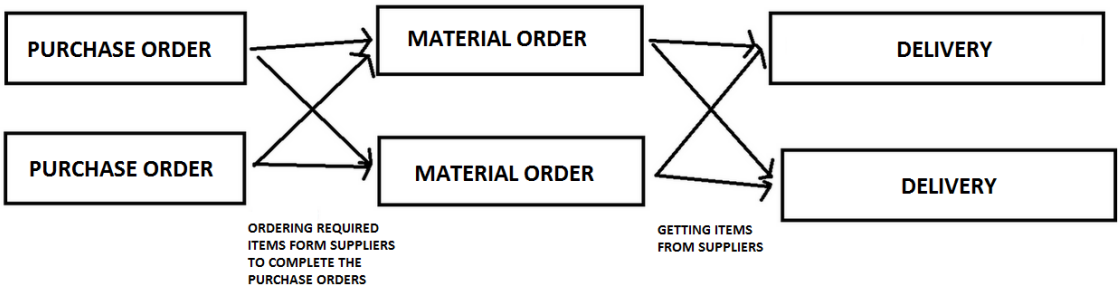

Figure 1.3. CD-shop artifact-centric model.

1.5. Artifact-Centric Service Interoperation Project

The Artifact-Centric Service Interoperation project started in June 2010. This European research project will end in May 2013 and has a budget of 4.7 million euros [3]. The goal of this project is to enable faster, smoother internet services to be provided by developing new techniques and tools. Process mining techniques are used to analyze the new systems and tools on which the project focuses [11]. Several universities are participating in the project, including the University of Tartu.

1.6. ProM, the Process Mining Framework

ProM [14] is a generic open-source Java framework for implementing process mining algorithms as plug-ins. ProM is developed by the Eindhoven Technical University and currently has two major editions: 5.2 and 6.0. Older ProM 5.2 features over 280 plug-ins for various process mining tasks.

While ProM 5.2 is successful, it has some limitations, most notably the tight integration between the tool and the GUI. Because of the limitations of ProM 5.2, the ProM 6.0 was introduced. ProM 6 is redesigned and developed from scratch and plug-ins for ProM 5.2 are incompatible with ProM 6. As Prom 6 was released in September 2010, there are still some plug-ins which are not ported to ProM 6. ProM is developed in Java, hence it supports multiple platforms, including Windows, Mac OS X and Linux. ProM already has dozens of plug-ins for both mining purposes (like control-flow mining or mining from less-structured processes) and analysis purposes (like verification, conformance and performance analysis). In this thesis, a plug-in for ProM 6 is being developed.

1.7. Relational Database Theory

A relation is a data structure which consists of a heading and an unordered set of tuples which share the same type. Relation is in first normal form (1NF) if every element of each tuple has atomic value, which means none of the element values can be decomposed.

A candidate key is a set of attributes from a particular relation, so that the relation does not have two distinct tuples with the same values for these attributes. Additionally the candidate key must be minimal, meaning there is no proper subset of these attributes which also forms a candidate key. From the set of candidate keys, a single unique key is selected and declared as the primary key for that data entity.

The candidate keys can be found by finding the minimal non-trivial functional dependencies. A set of attributes X in relation R is said to functionally determine another attribute Y also in R if and only if each X value is associated with precisely one Y value. If X functionally determines Y, then Y functionally depends on X. Minimal functional dependency means that if a set of attributes X determines an attribute Y, then there is no true subset of X, which also determines Y. Non-trivial functional dependency means that if a set of attributes X determines Y, then Y is not the element of set X.

If a set of attributes X in relation R functionally determines every other attribute Y in R and if the X is minimal, then X is a candidate key of this relation.

Id	**org:resource**	**order**	**quantity**	**supplier**	**cd artist**	**cd title**
489	CD Shop	9	1	bol.com	Sigur Ros	Takk
489	CD Shop	10	2	bol.com	Sigur Ros	Takk
489	CD Shop	11	1	amazon.com	Metallica	St Anger
489	CD Shop	12	1	bol.com	Rammstein	Mutter

Figure 1.4. Example of a relation.

In figure 1.4 we have a relation with the subsequent attributes: Id, org:resource, order, quantity, supplier, cd artist, cd title. This relation has 3 candidate keys: {order}, {quantity, cd artist} and {quantity, cd title}. {order} is called a simple key as it contains only one attribute while the other two are called composite keys for the opposite reasons. It

seems that {quantity, cd artist} and {quantity, cd title} are keys only because there is not enough data to prove otherwise.

2. CONTRIBUTION

2.1. Method Summary

The goal of this thesis is to develop and implement a method for discovering the important artifacts (or entities, to be more general) in the event logs and group event types into logical groups based on these artifacts. Later on (outside the scope of this thesis), the output of this implementation is used as an input for another algorithm, which discovers the relationships between the entities and so an entity-relationship model is built based on the event logs. The discovered model will then be used for conformance checking against artifact-centric business model or artifact life cycle discovery.

The event logs, as defined in section 1.3, can be represented as a relation in first normal form (1NF). Such relation contains a tuple for each event with attributes including timestamp, event type, resource and other data attributes in the log. To discover the entities from the event logs, the logs should be sorted into data sets according to the event types. We can do this by using selection. Every such data set can be seen as a database relation that holds all of the events of the same type.

We can then discover the candidate keys of the event type relations. A key in the relational database theory is an identifier, and thus it has to be unique. As the problem of the thesis is to group together the event types with the identical primary keys, the challenge is to find these keys. One relation can have several keys, all of these keys are called the candidate keys. In order to find the primary key for an event type, the program must first find the candidate keys. The candidate keys can be found in various ways. As the event logs can be represented as a relational database, it is possible to use the methods from the relational algebra on this data. One of the most efficient algorithms finds them via functional dependencies [12].

The functional dependencies are found using an algorithm called TANE [13]. After that, the keys are discovered from the dependencies. If a table has more than one candidate key, the user is asked to pick one as the primary key. The event types with the same primary keys can be grouped together to make an entity with a unique identifier. The entities are the candidates for the artifacts; the user can confirm or reject them if needed. The primary entities correspond to the artifacts, the secondary entities are associated with or shared by the artifacts. The identifiers of the secondary entities are composite and contain the identifiers of their associated primary entities.

The steps of the implementation of finding the entities is shown on figure 2.1. The implementation of the method is described in more detail in section 3.

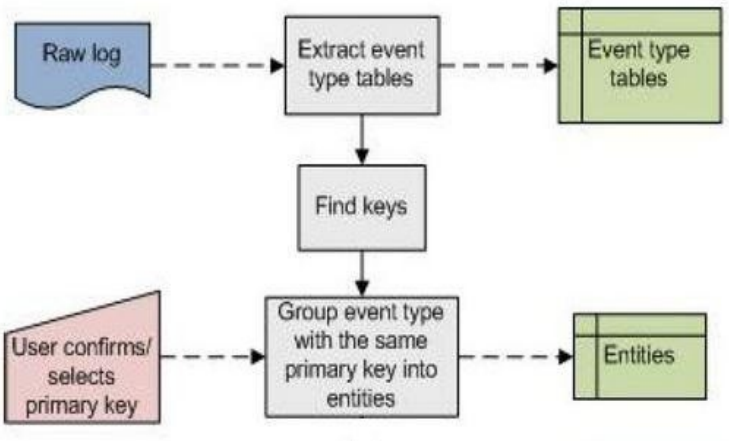

Figure 2.1. The process of discovering the underlying ER model for the raw logs [2].

2.2. Contribution of Implementation

From the implementational point of view the contribution of the thesis is as follows:

1. Modifying TANE to work with plain text or XML type of data instead of SQL database.
2. Implementing breadth-first search over the functional dependencies found by TANE to find the candidate keys.
3. Integration as a plug-in to industry standard process mining tool called ProM.
4. Graphical user interface for this implementation

There are also some specific requirements in order to integrate the method with the rest of the ACSI project. Most importantly, the ACSI uses ProM platform for process mining. As ProM itself is built on Java platform, the implementation of this thesis should also be in Java. It is also assumed that the event logs are either in XES or in MXML format and that event logs have a certain structure. All events in the logs should have an event type, timestamp and at least one data attribute. In this artifact-centric approach, the process instances are not observed separately but as continuous life cycles of multiple artifacts, i.e. the logs are structured as a single trace rather than in cases for separate process executions.

3. IMPLEMENTATION

3.1. About the Implementation

The implementation of finding the entities from the event logs is available in the appendix. This implementation can be divided into the following steps:

1. Integration with ProM.
2. Extracting the event type tables from the raw log input.
3. Finding functional dependencies from relational representation of event logs. The functional dependencies are found using an algorithm called TANE.
4. Finding the candidate keys from the functional dependencies. In case a relation has multiple candidate keys, the user is prompted to select one as primary key.
5. Grouping together the event types that have the same primary keys and integrating them into one entity.
6. The output is shown to the user or the entities are sent to another algorithm.

The implementation of finding the entities from the event logs is done in Java using Eclipse IDE. The advantages of using Java are:

1. A well-known process mining framework called ProM is implemented in Java, so a plug-in for ProM should also be implemented in Java.
2. An open-source implementation of TANE algorithm is available for Java [13].
3. Java is widely used for teaching programming in the University of Tartu and also for business programming in general.

3.2. Preparation of the Input Data

The first step prepares the process logs for further analysis. The plug-in (Figure 3.1) receives a set of logs from ProM as a Java data object. ProM itself supports various log formats, including XES and MXML. In this step, the events from the log are sorted into groups according to their event type attribute. For the example in figure 1.2, all of the events of type “Generate request” form one group, the events of type “Send quote” form another group and so on. Additionally, the TANE implementation that is being used gets the input by connecting itself to a relational SQL database and reading the relations directly from this database. Because ProM does not present event logs as a SQL database, it was necessary to alter the TANE implementation to accept the data as Java objects. When TANE has the input, it is ready to start processing it.

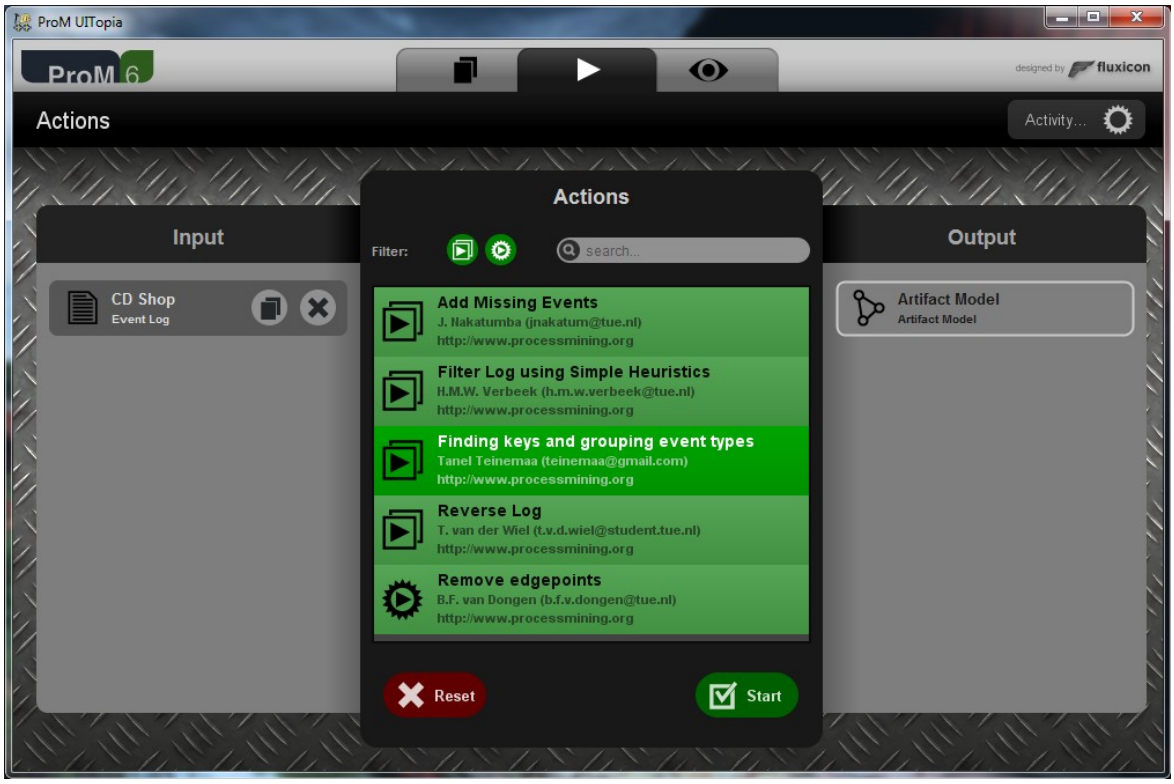

Figure 3.1. ProM plug-ins.

3.3. The TANE Algorithm for Finding the Functional Dependencies

In this step (Figure 3.2), functional dependencies for each group of event types are found. A sophisticated and efficient breadth-first search algorithm named TANE is used to find the functional dependencies.

TANE is based on partitioning the set of relation tuples, which makes the testing of validity of functional dependencies remain efficient for a large number of tuples. TANE's computing time complexity is $O(n*2^k)$, where k is the number of attributes, n is the number of tuples in the relation and n>>k. Despite the worst case exponential complexity of the number of attributes, the algorithm remains fast when the number of candidate keys in the relation is small.

One of the main advantage of this algorithm is the linear complexity of the number of tuples in the relation, while the other known algorithms have the complexity of O(n log n) or worse on the number of tuples [8].

Our implementation uses the Java implementation of the TANE algorithm created by Dr Jürgen Wäsch from the University of Hochschule Konstanz. Wäsch's implementation is open-source and under the GPLv2 license [13].

High-level pseudo code of the TANE implementation:

```
1 initialize()
2 while (Lx != empty)
3     compute_dependencies(Lx)
4     prune(Lx)
5     Lx+1 := generate_next_level(Lx)
6     x := x+1
```

Where level Lx is the collection of the attribute sets of size x = 1.

After all of the functional dependencies have been found, the new method of finding entities from the event logs continues by processing the functional dependencies in order to find the candidate keys.

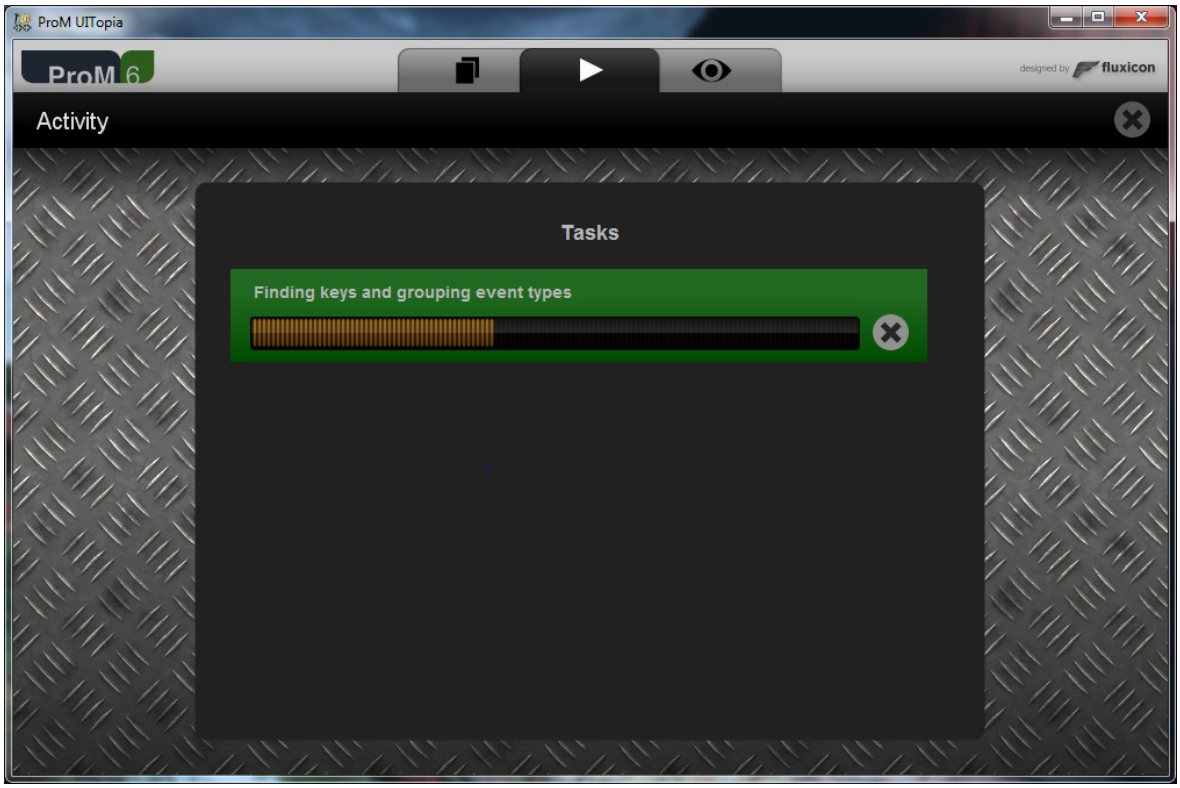

Figure 3.2. Processing data.

3.4 Finding the Keys

To find the candidate keys from the dependencies, a breadth-first search algorithm is used. This search has the exponential time complexity over the number of attributes in the relation. If necessary, the implementation can be improved to have polynomial time complexity by limiting the maximum number of attributes allowed for any candidate key [10].

High-level pseudo code of finding the candidate keys from the functional dependencies:

```
1 current_limit := 1
2 while (current_limit != maximum_limit)
3    recursive_generate_key(empty_key, 0, current_limit)
4    current_limit := current_limit+1
5 function recursive_generate_key(key, i, limit)
6    if(key->level = limit) check_key(key)
7    recursive_generate_key(key, i+1, limit)
8    recursive_generate_key(key->include_attribute(i),   i+1,
     limit)
```

When the candidate keys are found, it is time for the user to confirm them.

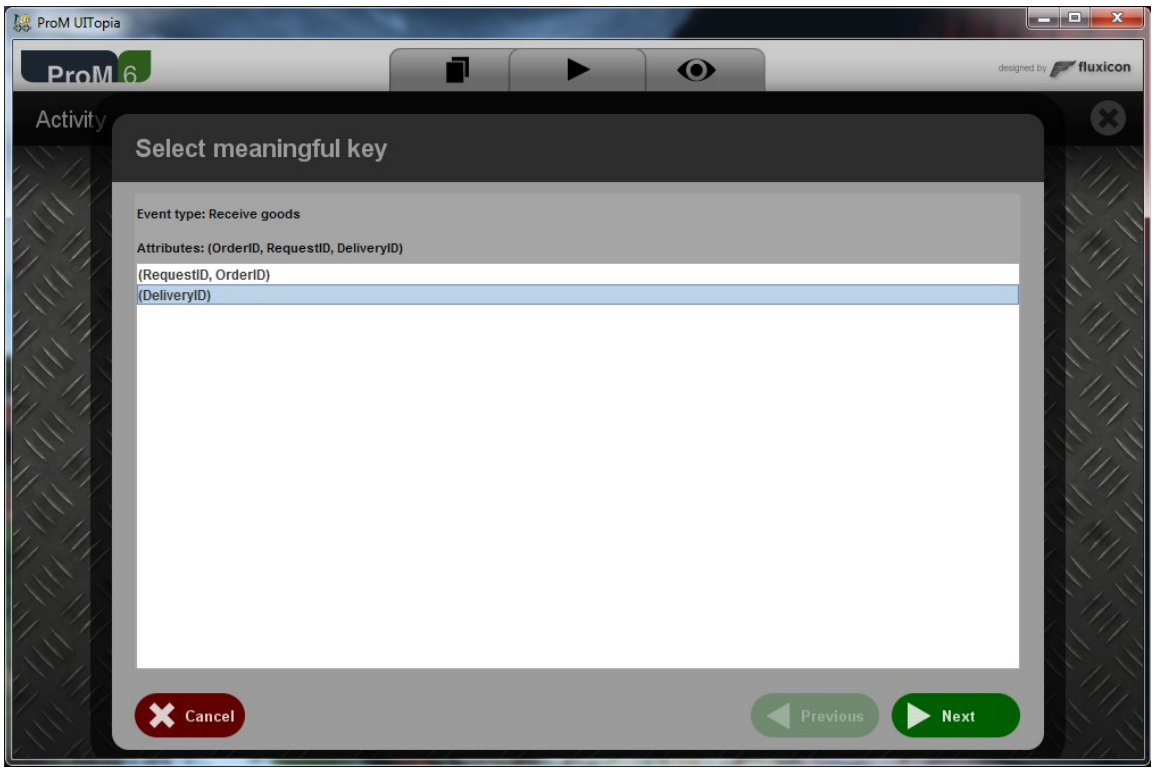

Figure 3.3. Selecting the identifiers.

3.5 The Output

Before the output can be shown to the user or subjected to further analysis, the user must pick exactly one candidate key for each event type to be the identifier for this event type entity (Figure 3.3). The majority of event types in the available example only have

one candidate key and in this case, user can only confirm it to be the identifier. But in some situations, for example when the event logs have too few data for a particular event, it is possible for an event type to have more than one candidate key. In this case, the user is responsible for picking the most meaningful identifier. As the identifier should be meaningful in the domain, there is no known way to algorithmically find this identifier out of all the candidate keys.

As the entities are now formed out of the event types and their identifiers, the method can send its outcome for further analysis. The result is also displayed to the user as a graphical representation of the entities in groups. (Figure 3.4).

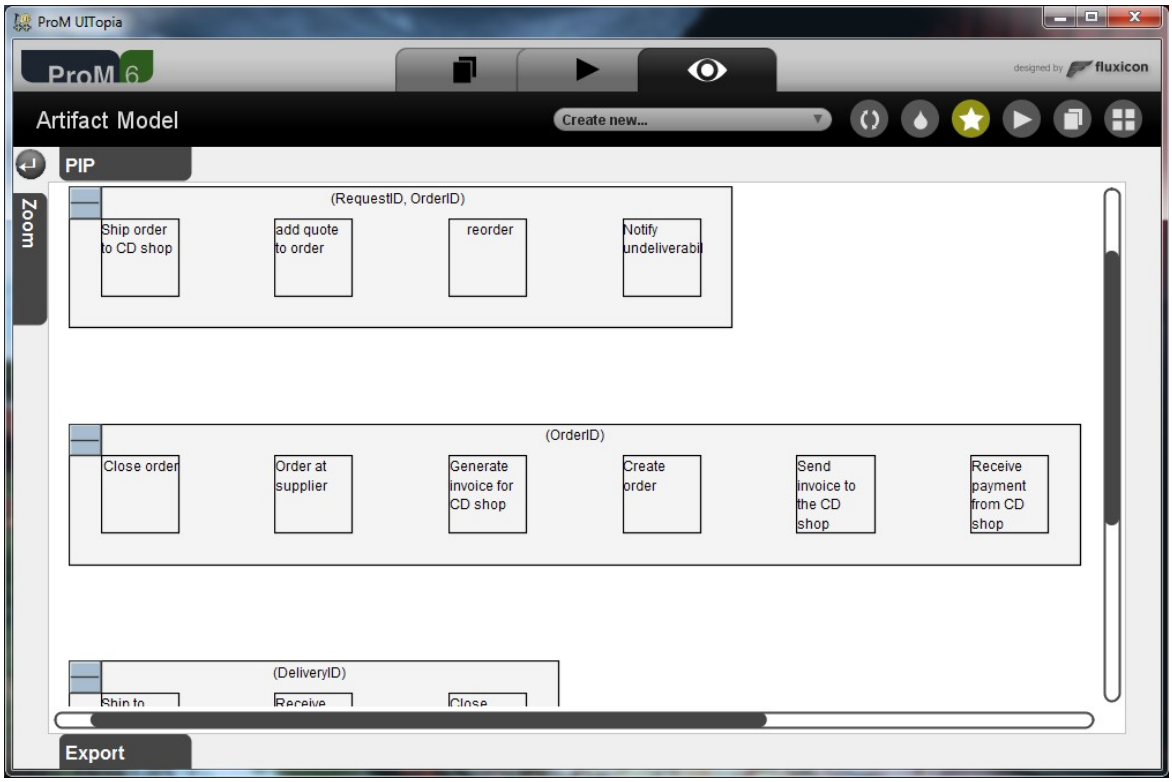

Figure 3.4. Grouped outcome.

4. EXPERIMENTS

4.1 Generating the Input Data

In the experiments the data is used from a simulation of the CD-shop example described in section 1.2. The steps followed to generate the data are presented in figure 4.1. Data that is used in the experiments is generated explicitly for process mining [4] and is not a part of this thesis.

Firstly, a business model is created by the domain expert. This model describes the behavior of the CD shop together with its environment consisting of customers and suppliers [2]. The model is implemented using CPN Tools [6]. CPN Tools is a tool for editing, simulating, and analyzing Colored Petri nets. Coloured Petri Nets is a graphical language for constructing models of concurrent systems and analysing their properties [5]. The fully automatic simulation in CPN Tools produces a textual file with a set of SQL statements to insert or update tuples in the database tables of the simulated artifact-centric system. These SQL statements are later executed in a batch, causing the database tables to be populated [2].

After that, the information in the database is serialized into a raw log using XeSAME. XeSAME is an application that provides a generic way for converting data from a data source to an event log [4]. The logs used in relation to this thesis are extracted in XES format, which enables their direct use in ProM.

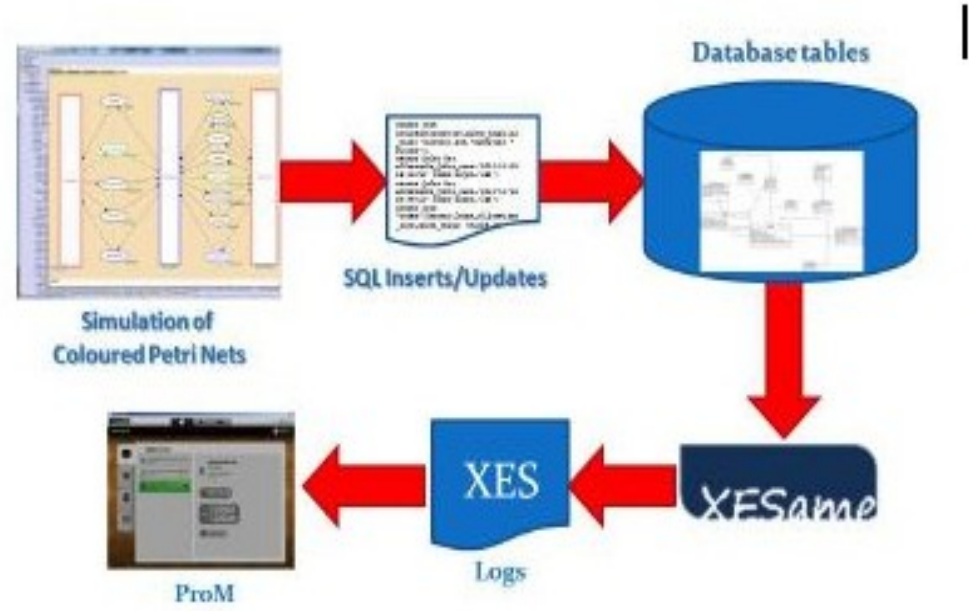

Figure 4.1. The steps followed to generate the data [2].

4.2 Testing

For testing, the method is applied separately on two log files that are generated by a simulation of the CD-shop example. The resulting entities are compared to the artifacts shown on the artifact models for the CD-shop. The artifact models are represented in the proclets notation based on Petri Nets [5]. Proclets are used as a light-weight formal model for artifacts. Proclets propose concepts for describing artifacts and their interactions [7].

The event types on the artifact models are not exactly the same as in the log files, so the results may be a bit different from the artifact models. It is also taken into account that each artifact can consist of several entities and some entities can be exclusively encapsulated in one artifact whereas other entities can be shared among the artifacts [2].

4.2.1 Single Material Order per Purchase Order

The log file is generated based on a model where the quotes are decomposed into orders to different suppliers. Items from different quotes are not combined together. Reordering if the item is unavailable is not allowed. The size of the log file is 148 KB and there are 528 events from 15 event types.

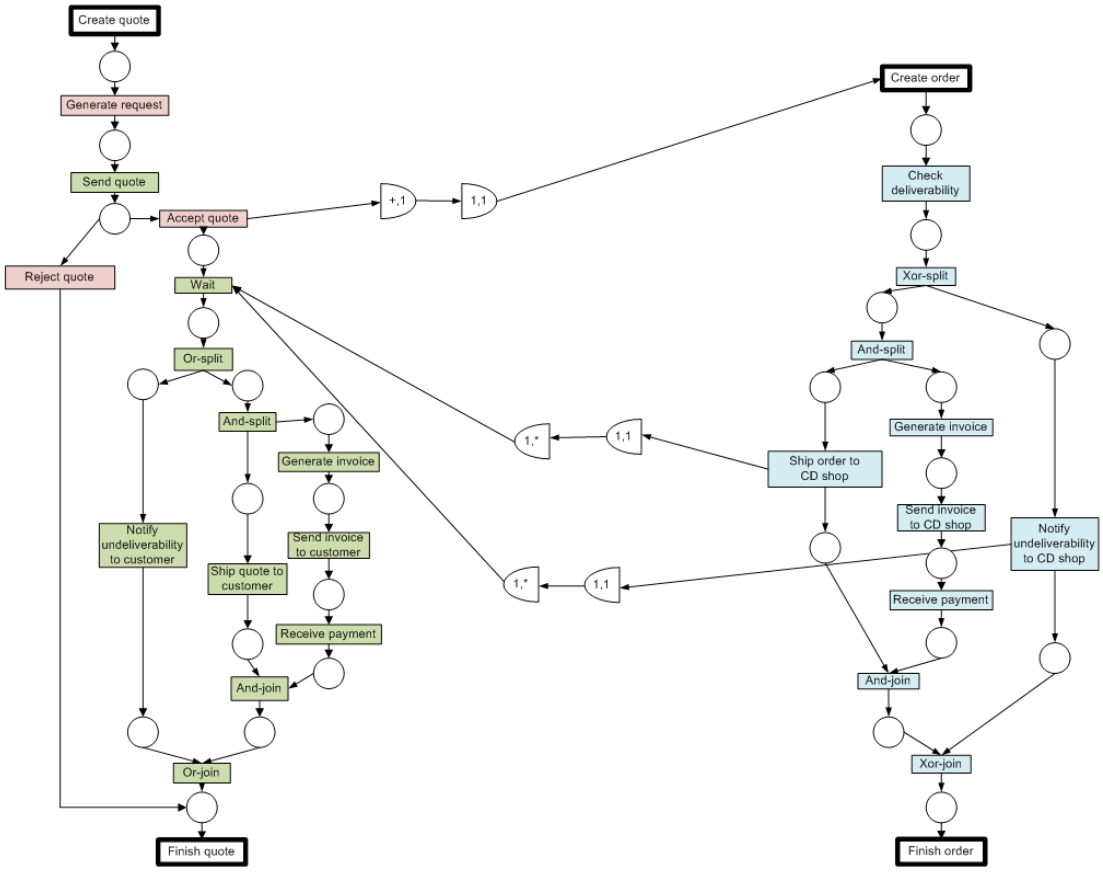

Figure 4.2. Artifact model 1.

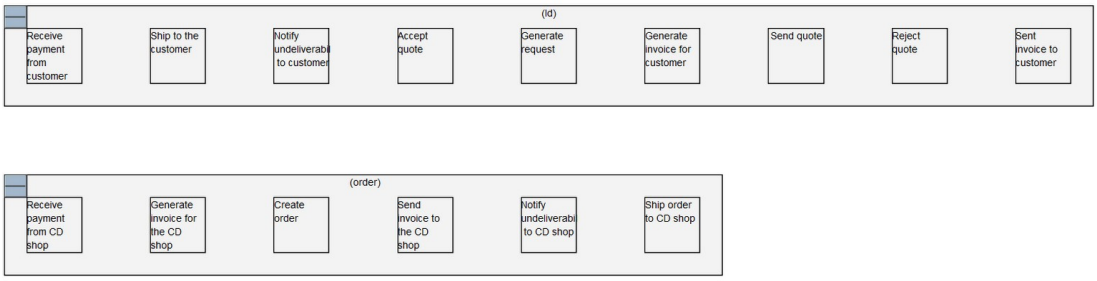

Figure 4.3. Entities from log file 1.

On the artifact model (figure 4.2) there are two artifacts: quote and order. The model in figure 4.2 uses some non-standard short-hand notation to represent AND- and OR-splits. These transitions were ignored in the analysis.

There are 11 event types in the quote artifact: create quote, generate request, send quote, accept quote, reject quote, generate invoice, send invoice to customer, ship quote to customer, receive payment, notify undeliverability to customer, finish quote.

There are 8 event types in the order artifact: create order, check deliverability, generate invoice, send invoice to CD shop, ship order to CD shop, notify undeliverability to CD shop, receive payment, finish order.

During the application of the method, the user was given the possibility to choose the primary key in case of the event type ‘notify undeliverability to cd shop’: {Id} or {order}. Using domain knowledge, the choice of {order} seems more meaningful. This event happens rarely in this log which is why {Id} is also found to be a key. In a larger sample this would not be the case since it can happen multiple times for the same Id.

The application of the method resulted in two entities (figure 4.3) with the discovered primary keys and event types:

1) Id: receive payment from customer, ship to the customer, notify undeliverability to customer, accept quote, generate request, generate invoice to customer, send quote, reject quote, send invoice to customer;

2) order: receive payment from CD shop, generate invoice for the CD shop, create order, send invoice to the CD shop, notify undeliverability to CD shop, ship order to CD shop.

All the event types of the Id entity are in the quote artifact. There are minor differences concerning the names of the event types, like receive payment - receive

payment from customer or ship quote to customer - ship to the customer, but the semantics of these names is evidently the same. There are two event types that are not a part of the first entity but are a part of the quote artifact. These are create quote and finish quote. The log file does not contain event types with these names. Therefore the absence of these event type is due to the difference between the event log and the artifact model, not due to a defect in the method.

All the event types of the order entity are in the order artifact. Similarly to the first entity-artifact pair, there are minor differences in the names of the event types, like generate invoice - generate invoice for the CD shop, but the semantics is the same. There are two event types that are not a part of the first entity but are a part of the order artifact. These are finish order and check deliverability. The log file does not contain event types with these names. Therefore the absence of these event type is due to the difference between the event log and the artifact model, not due to a defect in the method.

In the output the name of the event type ‘notify undeliverability’ is presented as ‘notify undeliverabil’, because the word ‘undeliverability’ is too long. This could be a minor issue with the method but as the main purpose of the method is to send the entities to the next algorithm (for conformance checking) and not to present them to the user, it is not an important issue.

In conclusion, the method worked correctly in case of the first event log.

4.2.2 Multiple Material Orders per Purchase Order with Reorders

The log file is generated based on a model where reordering is allowed and items for the same supplier from different quotes are combined. The size of the log file is 944 KB and there are 4376 events from 22 event types.

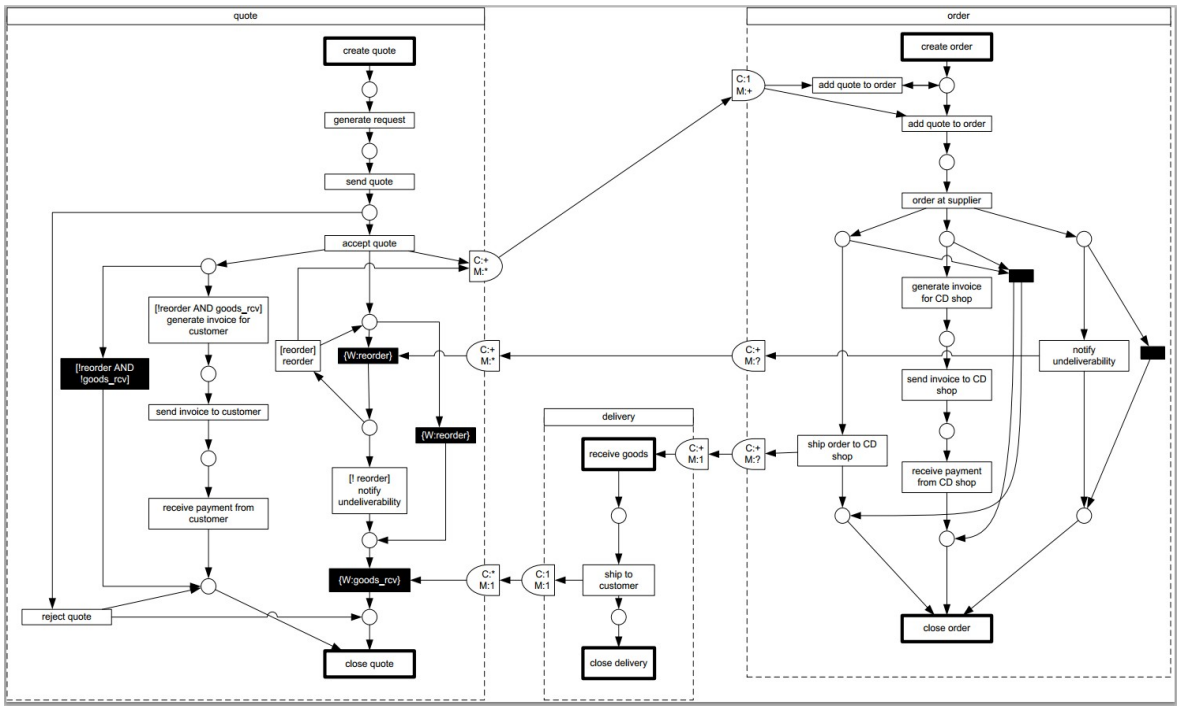

Figure 4.4. Artifact model 2.

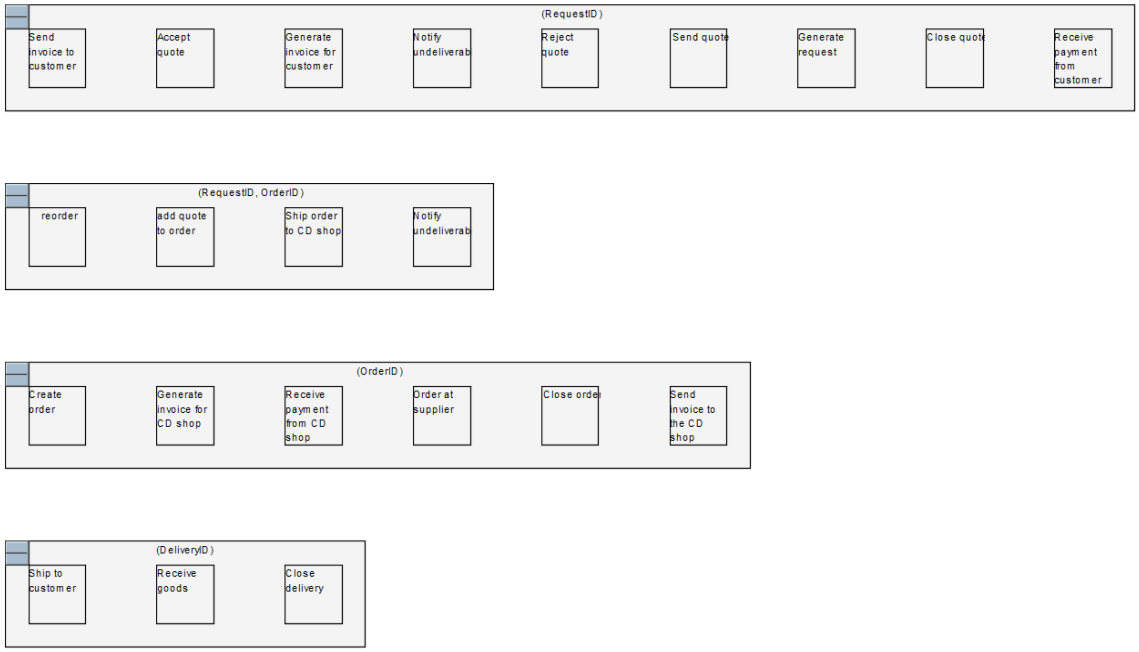

Figure 4.5. Entities from log file 2.

On the artifact model (figure 4.4) there are three artifacts: quote, delivery and order.

There are 11 event types in the quote artifact: create quote, generate request, send quote, accept quote, generate invoice for customer, send invoice to customer, receive payment from customer, reorder, notify undeliverability, reject quote, close quote.

There are 3 event types in the delivery artifact: receive goods, ship to customer, close delivery.

There are 9 event types in the order artifact: create order, add quote to order, add quote to order, order at supplier, generate invoice for CD shop, receive payment from CD shop, ship order to CD shop, notify undeliverability, close order.

During the application of the method, the user was given the possibility to choose the primary key in case of the event type 'receive goods'. Possible keys to choose from are composite key {RequestID, OrderID} and simple key {DeliveryID}. Using domain knowledge, the choice of {DeliveryID} seems more meaningful.

The application of the method resulted in four entities (figure 4.5) with the subsequent primary keys and event types:

1) RequestID: send invoice to customer, accept quote, generate invoice to customer, notify undeliverability, reject quote, send quote, generate request, close quote, receive payment from customer;

2) RequestID, OrderID: reorder, add quote to order, ship order to CD shop, notify undeliverability;

3) OrderID: create order, generate invoice for CD shop, receive payment from CD shop, order at supplier, close order, send invoice to the CD shop;

4) DeliveryID: ship to customer, receive goods, close delivery.

All the event types of the RequestID entity are in the quote artifact. There are two event types that are not a part of this first entity but are a part of the quote artifact. These are create quote and reorder. In fact, the log file does not contain an event type of 'create quote'. Therefore the absence of this event type is due to the difference between the event log and the artifact model, not due to a defect in the method. The reorder event type is included in the second entity. As mentioned above, the artifact may consist of several entities. There exist the subsequent entries in the event log:

There exist the following entries in the event log (figure 4.6).

```
<event>
     <string key="OrderID" value="22"/>
     <string key="RequestID" value="488"/>
     <string key="concept:name" value="reorder"/>
     <date key="time:timestamp"
          value="2011-06-05T20:35:36.000+02:00"/>
</event>
<event>
     <string key="OrderID" value="22"/>
     <string key="RequestID" value="518"/>
     <string key="concept:name" value="reorder"/>
     <date key="time:timestamp"
          value="2011-06-06T04:39:56.000+02:00"/>
</event>
<event>
     <string key="OrderID" value="28"/>
     <string key="RequestID" value="488"/>
     <string key="concept:name" value="reorder"/>
     <date key="time:timestamp"
          value="2011-06-07T21:44:25.000+02:00"/>
</event>
```

Figure 4.6. An example of three reorder events in the log file.

In all of these entries the event type is reorder. In the first and second entries the OrderID is the same but the RequestID is different, therefore the OrderID does not determine the RequestID. In the first and third entries the RequestID is the same but the OrderID is different, thus the RequestID does not determine the OrderID. Hence, the method worked correctly in positioning the reorder event type to a separate entity with the key of {RequestID, OrderID} and not positioning it to the entity with the key RequestID, as would seem appropriate according to the artifact model.

All the event types of the OrderID entity are in the order artifact. There are three event types that are not a part of the third entity but are a part of the order artifact. These are add quote to order, notify undeliverability and ship order to CD shop. All of these event types are included in the second entity. It is allowed that one artifact consists of several entities. One event type in the second entity (reorder) is a part of the quote artifact and three event types in the second entity are a part of the third entity. This is also allowed, because an entity may be shared among several artifacts.

The fourth entity contains exactly the same event types as the delivery artifact.

In consequence, the method worked correctly on the second event log. The first, third and fourth entities are primary entities. The second entity is a secondary entity.

To sum up, the method worked as expected on the event log files. As it was in the first section, a minor disadvantage in the graphical interface was discovered that in case of long words the word might not be properly presented on the screen. However, as the main purpose of the method is to send the entities to another algorithm and not to show them as output, this is not a major problem.

CONCLUSION

The purpose of this thesis was to present a new method for discovering entities and candidate artifacts in the event logs. The method was implemented in Java as a plug-in for ProM. The method discovers the relevant entities and the candidate artifacts in the event logs and groups the event types into logical groups based on these entities.

As the method was implemented as a part of the ProM framework, it takes as input logs in a specific XML event log format called XES. Then the implementation serves this information to the sophisticated TANE algorithm, which finds the functional dependencies for each event type. After finding the dependencies, breadth-first search was used to find the candidate keys from the functional dependencies. In case there were multiple candidate keys for the same event type, the user is allowed to pick one of them. In the end, the event types with the same keys are grouped together and the result is visualized using the ProM built-in user interface.

The method was tested on two event logs generated by a simulation of the CD-shop example. The method worked correctly on these log files and produced the expected entities. A minor problem was discovered that in case of long words in the names of the event types the word might not be properly presented in the output.

The output of this algorithm can be used as an input for another algorithm, which discovers the relationships between the entities and so an entity-relationship model is built based on the event logs. This model can be used together with the event logs for conformance checking between the logs and an artifact-centric model or to discover artifact life cycle.

Olemite leidmine protsessi läbiviimise logidest

Bakalaureusetöö (6 EAP)

Tanel Teinemaa

Lühikokkuvõte

Töö on kirjutatud protsessikaeve valdkonnas Artefaktikeskse teenuste koosvõime projekti (ACSI) raames. Töö eesmärgiks oli luua meetod sündmuste logidest olemite avastamiseks ja seda meetodit rakendada.

Loodud meetod on kirjutatud Javas ning kujutab endast pluginat ProM raamistikule. ProM on geneeriline avatud lähtekoodiga Java raamistik protsessikaeve algoritmide rakendamiseks pluginatena.

Olemite leidmise protsessi saab jaotada järgmisteks sammudeks:

1. Integreerimine ProM-iga.
2. Sisendandmetest (XES formaadis logifailidest) sündmuste tüüpide relatsioonide koostamine.
3. Funktsionaalsete sõltuvuste leidmine sündmuste logide relatsioonilisest esitusest. Funktsionaalsete sõltuvuste leidmiseks kasutatakse algoritmi TANE.
4. Funktsionaalsete sõltuvuste alusel kandidaatvõtmete leidmine. Kui relatsioonil on mitu kandidaatvõtit, palutakse kasutajal valida neist üks primaarseks võtmeks.
5. Sama primaarse võtmega sündmustest moodustatakse üks olem.
6. Kasutajale esitatakse töö käigus moodustatud olemid väljundina või saadetakse need järgmisele algoritmile töötlemiseks.

Meetodit testiti kahe logifaili puhul, milles olid andmed CD-poe näitel. Tulemuseks saadud olemeid võrreldi vastavate skeemidega (joonised 4.2 kuni 4.5). Meetod töötas mõlema logifaili puhul korrektselt.

REFERENCES

[1] W.M.P. van der Aalst, H.A. Reijers , A.J.M.M. Weijters , B.F. van Dongen, A.K. Alves de Medeiros, M. Song, H.M.W. Verbeek. Business process mining: An industrial application.
Information Systems, July 2007, Volume 32, Issue 5, Pages 713–732.

[2] ACSI - Artifact-Centric Service Interoperation. Deliverable 3.1.

[3] ACSI Factsheet. http://www.acsi-project.eu/papers/factsheet-final.pdf. Last visited: 8.05.2012.

[4] J.C.A.M. Buijs. Mapping Data Sources to XES in a Generic Way. March 2010

[5] Coloured Petri-Nets. http://cs.au.dk/CPnets/. Last visited: 8.05.2012.

[6] CPN Tools. http://cpntools.org/. Last visited: 8.05.2012.

[7] Dirk Fahland, Massimiliano de Leoni, Boudewijn F. van Dongen, and Wil M.P. van der Aalst. Behavioral Conformance of Artifact-Centric Process Models. Business Information Systems. Lecture Notes in Business Information Processing, 2011, Volume 87, Part 2, Pages 37-49.

[8] Y. Huhtaka, J. Kärkkäinen, P. Porkka and H. Toivonen. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. The Computer Journal, 1999, Volume 42, Issue 2, Pages 100-111.

[9] Oxford Dictionaries. http://oxforddictionaries.com/. Last visited: 8.05.2012.

[10] M. Kurant, A. Markopoulou, P. Thiran. On the bias of BFS. International Teletraffic Congress (ITC 22), 2010.

[11] More efficient, lower-cost internet services through research by Mathematics and Computer Science department.
http://www.processmining.org/blogs/news/more_efficient_lower-cost_internet_services_through_research_by_mathematics_and_computer_science_department. Last visited: 8.05.2012.

[12] H. Saiedian, T, Soencer. An Efficient Algorithm to Compute the Candidate Keys of a Relational Database Schema. The Computer Journal, 1996, Volume 39, Issue 2, Pages 124-132.

[13] TANE-java. http://www-home.htwg-konstanz.de/~waesch/DBResearch/content/tane_java.htm. Last visited: 8.05.2012.

[14] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, W.M.P. van der Aalst. ProM 6: The Process Mining Toolkit. Industrial Engineering, 2009, Volume: 489. Publisher: CEUR-WS.org, Pages: 1–4.

[15] ACSI Description of Work Part B. http://www.acsi-project.eu/papers/DoWPartB-tech.pdf. Last visited: 8.05.2012.

[16] D. Cohn, R. Hull. Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. IEEE Data Eng. Bull., 2009, 32:3–9.

APPENDIX

The attached optical disc contains the source code of the implementation and the ProM environment to run it. It also contains a readme.txt explaining how to run the the implementation.