UNIVERSITY OF TARTU
Faculty of Mathematics and Computer Science
Institute of Computer Science

Riivo Kikas

Discovering mapping between artifact-centric business process models and execution logs

Master's Thesis (30 ECTS)

Supervisor: Viara Nikolaeva Popova, PhD

Author: ..... "....." June 2011

Supervisor: ..... "....." June 2011

Approved for defence:

Professor: .... "....." June 2011

TARTU 2011

Contents

Chapter 1

Introduction

Information systems enable better business operation by supporting business processes with execution and monitoring of workflows. Workflow management systems enforce that all the tasks in the processes are completed, uncompleted tasks are followed up and provides resource management to allocate resources for different tasks. Most worfklows are designed in a process-oriented view using notations such BPMN or UML activity diagrams. This way is not natural for the actual processes as it separates business data objects from the process model.

In recent years, a new approach, artifact-centric modelling has emerged to model processes, concentrating on business artifacts and their interactions. This enables to model systems using business objects and their interactions, at the same time keeping data attributes with each object. Existing process mining techniques consider only process-centric models, but process mining tasks, such as conformance checking need to be carried out also on artifact centric models. One prerequisite for conformance checking is the presence of mapping, which assigns the events in the logs to activities in the model.

1.1 Motivation

Workflow systems executing business processes produce execution logs of activities carried out, that can be used for process mining and diagnostics. It may happen, that the names and attributes in the model do not correspond exactly to the names of the events in the execution logs.

Mapping between the elements in logs and models is needed in order to carry out process analysis, such as conformance checking between the model and the log. Such mapping can be provided by hand or found automatically. Latter approach is easier for users, as models can be complex and contain thousands of tasks.

1.2 Problem statement

Given an execution log and an artifact-centric process model, we are interested in finding the mapping between process activities in the model and events in the log. Finding the mapping is not trivial, as there are various possibilities for deviations between models and logs: the labels might not be syntactically or semantically similar and there can be structural differences. These differences may be present for example when an older version of the model is used. As a result of all the possible modifications, there might be no perfect mapping and finding the mapping automatically facilitates end users work.

The goal of this thesis is to discover such mapping based only on the data present in the logs and models. Different methods are studied in order to extract the mapping and automate the task. This thesis does not aim to provide out of the box solution, it is more of a study of the mapping discovery task in artifact-centric process models.

The proposed method uses Proclets[24] and Petri nets[19] to model artifacts and their life-cycle. Behavioral profiles are used to transform logs and models into graph structures and to enable comparison. To measure similarity between activity labels, we use syntactic and semantic similarity measures. To construct the mapping, Similarity Flooding algorithm [18] and a greedy algorithm, that minimises graph edit distance, are used.

Proposed method is also suitable for process-oriented cases as finding the mapping between an artifact and its execution logs is the same as using only a single process.

1.3 Contributions

Main contributions of this thesis are:

- Description of a method for extracting mapping between events in log files and activities in artifacts.
- Experimental evaluation of the method based on a series of case studies with different model and log pairs.

1.4 Structure of the thesis

The rest of this thesis is structured as following:

- In **Chapter 2** we give an overview of the necessary background to understand the context, including artifact centric process mining.
- In **Chapter 3** we describe the related methods and algorithms that are used in the mapping discovery.

- In **Chapter 4** we describe the method for discovering a mapping.
- In **Chapter 5** we carry out series of experiments to assess the quality of the method.
- in **Chapter 6** we provide conclusion of thesis and discuss possible future work.

Chapter 2

Background

In this chapter we introduce the concepts of workflows, business processes, artifact-centric modeling of processes and process mining. The running example of for this thesis is also described.

2.1 Workflow systems

Organisations perform activities that are designed to achieve the goal or purpose of the organisation. The collection of such activities and their relationships can be considered as a process and we call this a business process. Nowadays information systems need to support business processes to enable smooth business operation throughout organisation and provide other aspects such as controlling and monitoring processes.

A workflow [1] can be defined as a collection of tasks organized to accomplish some business process. Workflows are case-oriented, i.e., tasks are executed for specific cases. Loan applications and insurance claim handling are typical case-driven processes, where one case can describe the handling of one application or claim. A task in a workflow system may be performed manually or by a software system. Human tasks may include working with the system, for example entering data. Examples of tasks might include generating an invoice or updating a record in a database.

For example in banking, handling loan applications is one of the core business processes of the bank, resulting in earnings from interest rates. There are specific rules, roles and activities in the process. A new case starts usually with application from the customer with his details and income history and at some point contains credit check done by the bank.

The main purpose of a workflow management system is the support of the definition, execution, registration and control of processes [25]. Workflow systems offer logistical operations to support business processes, ensuring that the proper activities are executed by the right person. The system can be configured to log all activities performed during

a workflow case execution. The produced logs can be used in process analysis, providing information about each task execution.

2.2 Petri nets

A Petri net [19] is a formal modeling method often used to represent processes and in particular business processes. A Petri net is a directed bipartite graph, composed of two types of nodes: places and transitions. Nodes are connected with each other using directed arcs. Arcs can only connect a transition to a place or a place to a transition, connections between nodes of the same type are not allowed. In the graphical representation, places are denoted by circles and transitions by rectangles.

Definition 2.1 (Petri net)**.** *A Petri net is a triple* (P, T, F)*:*

- P *is a finite set of places*
- T *is a finite set of transitions,* $P \cap T = \varnothing$
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs*

A place can contain tokens, graphically denoted as black dots. The transition is *enabled* as soon as all of its input places, that is places connected via incoming arcs, contain a token. An enabled transition may *fire*, consuming a token from each of its input places and produces a token for each of its output places. The *marking* of a net is a distribution of tokens over the set of all places. A *system* $S = (N, m)$ is given by a Petri net N and an initial marking m. The set of all reachable markings of S is denoted by $[N, m\rangle$. A *firing sequence* $\sigma : \{0, \ldots, n-1\} \to T$ of length n specifies a sequence of transitions that can be fired in sequential order, resulting in a new marking.

Petri nets are used for several reasons in workflow modeling, especially for their formal semantics. A process specified with Petri net, has a precise, mathematically formal definition. Moreover, Petri nets support all operations needed to model a workflow process. Due to the wide usage of Petri nets in different domains such as model checking and system simulation, they have been studied extensively and their mathematical foundation allow analysis of the processes.

Mapping workflow management concepts to Petri nets

When dealing with processes, we are interested in the execution of activities. A Petri net can be used such that all activities are represented by transitions and firing a transition means execution of a task.

When dealing with Petri nets in the process domains, a subclass of Petri nets called workflow-net (WF-net) [25] is typically used.

Definition 2.2 (Workflow net)**.** *A Petri net $PN = (P, T, F)$ is a WF-net if and only if:*

- *It has two special places i and o. Place i is a source place and there are no incoming arcs to i. Place o is sink place and has no outgoing arcs.*
- *Adding a transition t to PN that connects place o with i, results in a strongly connected net.*

A WF-net can be used to describe some process model, with its fixed start and end place and combined with *soundness* property, that guarantees that there is a path to the end place, i.e., the process can terminate. More properties of WF-nets and analysis is given in [25].

2.3 Process mining

Workflow management systems record information about the execution of the activities which are stored in log files. The event logging usually is present due to requirements for information systems to preserve data about history or maintain audit trail. Event logs contain information about events, that refer to activities performed during the workflow and also to the process instance that the event was associated to. Each log entry typically has a timestamp attribute, indicating the precise time of the event. Additional information may also be present, such as the data used in the activity and the person executing the task. For example, when a business process is implemented using Petri nets, a log entry can be produced when a firing of a transition occurs in the model.

Availability of such data enables to gather more information about the processes. Process mining [26] is the extraction of information about processes from event logs. The aim is to use the data from the logs to obtain more information about the processes, discovering the process model, the social structures of the organisation or providing additional information about the process. Three different types of process mining can be conducted: discovery, conformance checking and enhancement [23].

A discovery technique summarises the behavior stored in the event log files and produces a process model without using any information. By analyzing the logs, a Petri can be constructed based only on the data in the log files and not seeing the actual implementation of the system [33]. Typically there is no known model of the process present and discovery provides methods to obtain it based on the data.

The second task of process mining, conformance checking [22], dealing with measuring how well the log conforms to the known model. Deviation from the model may occur during system implementation, i.e, the model specifies that a security check must be carried out before handling specific tasks, but in practise security check is not done. Also, workers may deviate from the process by deciding not to execute some tasks to

better serve the customers and therefore indicating there is a better way to carry out the process that specified in the model. When an existing documented process model is present, conformance checking helps organisations to discover such deviations and provide reasoning for the causes.

Fitness is a measure of how well the known Petri net fits with logs. A naive version of fitness can be considered as the percentage of cases in logs that the Petri net can replay, i.e., if the Petri net cannot fire some transition of the stored log, it is regarded as it can not be replayed and does not fit. More formally, fitness keeps track of the number of missing or artificially generate tokens during the replay. More information about it can be found in [23]. It is shown that a Petri net can be constructed to parse almost every event log [22] and therefore other methods are also used to measure conformance such as *appropriateness*, which represents the degree of accuracy with which the process model describes the observed behavior, combined with the degree of clarity in which it is represented [22]. A prerequisite for the conformance checking is to provide a mapping that specifies which events in logs correspond to which transition in the model.

The third task of process mining, enhancement, deals with providing information about how to restructure the process. Timestamps in the event logs can be used to study the presence of bottlenecks and the process can be re-engineered to remove these conditions.

The ProM framework [27] is a toolkit providing common process mining methods for discovery, conformance checking enchantment.

2.4 Artifact-centric modeling

Classical process modeling methods consider different processes as independent instances that are executed in isolation. However, in real world, processes interact with each other exchanging business data. The artifact-centric modeling [4] approach is one way to describe complex inter- and intra-organizational processes in a modular way [12].

An artifact is an object that participates in the process. Examples of artifacts are an electronic order, a paper form or a delivery package. These objects have data attributes such as fields of the order from, specific order contents and from whom this order originates. Different artifacts have relations between them, i.e., a delivery package is created after processing some order.

Each artifact has its own life-cycle that describes the states and possible transitions of the object (for example, an order gets started, approved and delivered) and an information models for holding associated data. The idea of artifact-centric approach is to model each artifact separately and the interactions between artifacts. A process model in this context emerges automatically from artifact interactions.

A process execution creates new instances of artifacts, denoting specific objects such

as *customer order number 1* and *customer order number 2*. The attributes of instances are changed during the execution. The states of artifacts also change during the execution, following the lifecycle model of the specific artifact.

The artifact-centric approach provides better ways to model inter-organizational processes and concentrates on business artifacts, that provides a more natural way to model business operations. Advantages over the traditional processes oriented approaches are that artifacts make the data as well as the process visible.

For artifact-centric processes, there does not exist a unique notion of a case or process instance, as the process cannot be considered in isolation. This raises requirements for new modeling methods that support process executions where several different cases overlap and synchronize at different points.

2.4.1 CD shop example

As a facilitating example throughout this work we are using a process of an online CD shop, also used in [11, 10] as an example. The on-line CD shop offers customers to order CD-s from large catalogues, originating from different suppliers. The process starts with a request for CD-s from the customer. The shop sends a quote for CD-s to the customer and, when the customer accepts the quote, it is split into several orders, one per each CD supplier. Each order again contains all quotes for CD-s to the same supplier. In case some CD-s are not available at the supplier, the CD shop is notified and it forwards the information to the customer. Invoicing is also part of the process, as suppliers issue invoice to the shop and the shop in turn to the customers, and both parties expect payment for the invoice. This example focuses on the back-end of CD shop, interaction with the customer is not extensively modelled.

The underlying data model (Figure 2.1) of the process contains information about all the objects that are used in the process. Relevant information is stored in the data attributes and relations between the objects, stating how many of the objects of one type are related to how many different types.

In this CD shop example, we consider two business artifacts: a quote and an order. All other related information specified in the data model is encapsulated in the artifact types and in their interactions.

2.4.2 Proclets

Proclets provide methods for modeling artifact-centric processes by defining artifacts and their interactions [24]. A *proclet* $P = (N, ports)$ consists of a labeled Petri net N, which describes an internal lifecycle of an artifact and a set of ports, through which P can communicate with other proclets. The whole system can be described as a *proclet system*

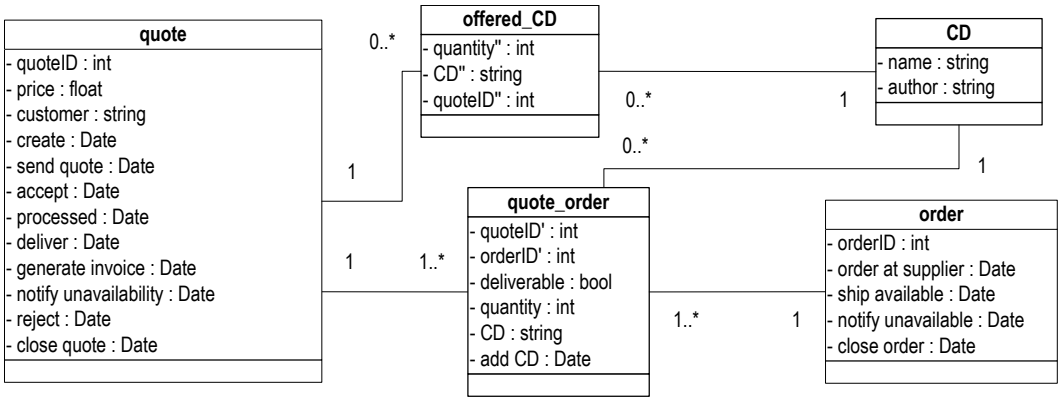


Figure 2.1: The CD shop data model [11].

$\mathscr{P} = (\{P_1, \ldots, P_n\}, C)$ consisting of a set of proclets and a set C, of channels between proclets. Each channel has two ports, connecting two proclets. Proclets can communicate through the channels, by sending and receiving messages.

In this example we have two artifacts: *Quote* and *Order* and for each artifact we have a proclet (Figure 2.2).

An important part of proclets is the interactions between artifacts. The ports, denoted by half-round shape, connect proclets using the channels, denoted as dashed lines between ports. The symbols 1, ?,+ on port shape specify its properties. The first annotation, called cardinality, states how many messages one proclet instance sends to or receives from other instances when the transition occurs. The other annotation, multiplicity, states how many times this port can be opened during the life cycle of the model. For example, the port connected to *accept* transition has cardinality "*+*" meaning that the port can send multiple messages at a time, that is the information about multiple CD-s. The multiplicity is "*1*", meaning that the port can be opened only once, i.e., multiple CD-s must be accepted and sent for ordering at the same time.

A proclet model concentrates only on the process aspects of artifacts and interactions between them. It does not incorporate the data model. However, it can easily be extended since various extensions of Petri net for dealing with data exists, such as Coloured Petri Nets [16].

2.4.3 The artifact conformance checking problem

The lack of unique process instances creates the necessity for different conformance checking methods. The idea is still the same, that is to measure how well the execution corresponds to the known model or if the given event log can be replayed by the supplied Proclet system [10]. But instead of single process instance, there is now need to consider conformance of different artifacts and their interactions.

Artifact-centric conformance checking introduces several aspects to the conformance

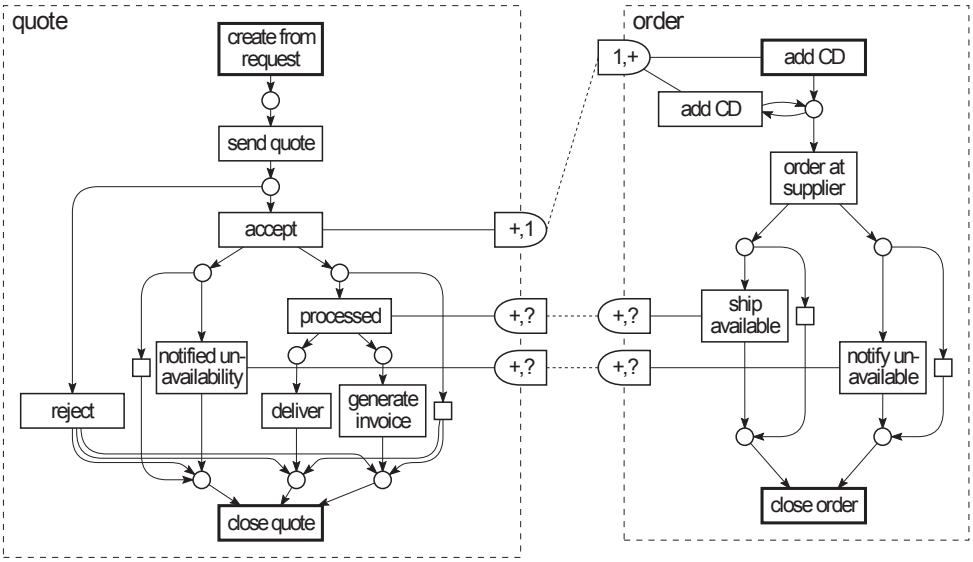


Figure 2.2: The CD shop as proclet model [11].

checking problem [28]:

- **Behavioral Conformance**. Conformance between the behavior of the instances of each artifact and the specified life cycles of the corresponding artifacts. This is the most straightforward adoption of the classical conformance notions to a single artifact. Conformance of each artifact is considered in isolation and the interaction is discarded.

- **Interaction conformance**. Measures how well the interaction between artifacts conforms to the structure. This involves deciding whether communication links are correspond with the model and the properties of the ports match to the model.

- **Data conformance**. Behavioral and interaction conformance cover the structure and life-cycle of the model, but do not consider how the underlying data is updated. Data conformance measures how the decisions in the model conform to the specification.

- **Structural Conformance**. Measures conformance between the overall structure of logs and models. Same system can me modeled in different ways, for example representing with different number of artifacts and their interactions.

The methods in this thesis are important for behavioral and interaction conformance. We are extracting the mapping between activities in the Proclet model and in the event log, that enables to replay each artifact separately and also extract interaction between artifacts.

Chapter 3

Related work

In this chapter we describe related concepts that are used in the proposed method to the mapping discovery problem. Our method utilizes behavioral profiles as a base for the representation of the process model and logs. To find the mapping, graph matching is used between the behavioral profiles of process models and execution logs. Different label similarity measures are considered for finding similar activities in the graphs.

3.1 Behavioral profiles

The behavioral profile of a process model captures the behavioral aspects of a process, such as mutual exclusion of activities or potential occurrence order for a pair of activities. Behavioral profiles were originally proposed for process model alignment and measuring consistency between corresponding models [31]

Advantages of using behavioral profiles are that they enable to capture the underlying behaviour of the process in a compact way and we can discard the original process modeling notation and depend only on whether the process can be represented using Petri nets. The behavioral profile has been shown to be less sensitive to process model projection [31].

A behavioral profile consists of three relations, specifying whether two activities might happen in a strict order, exclusively or in an interleaving order. All behavioral relations depend on the concept of a weak order.

Definition 3.1 (Weak Order Relation)**.** *Let* $(N, [i])$ *be a WF-system. The weak order relation* $\succ \subseteq T \times T$ *contains all pairs* (x, y)*, such that there exists a firing sequence* $\sigma = t_1, \ldots, t_n$ *with* $(N, [i])[\sigma\rangle$*,* $j \in 1, \ldots, n-1$*, and* $j < k \leq n$ *for which holds* $t_j = x$ *and* $t_k = y$*.*

The weak order relation $x \succ y$ between two transitions in the net specifies that firing of x happens before y in at least one possible execution, but does not have to occur directly before y and other transitions may be fired between the two activities.

Definition 3.2 (Strict Order Relation). *Let $(N,[i])$ be a WF-system. The strict order relation $\rightsquigarrow \subseteq T \times T$ contains all pairs (x,y) such that $x \succ y$ and $y \not\succ x$.*

Definition 3.3 (Exclusiveness Relation). *Let $(N,[i])$ be a WF-system, with the Petri net N and initial marking $[i]$. The exclusiveness relation $+ \subseteq T \times T$ contains all pairs (x,y) such that $x \not\succ y$ and $y \not\succ x$.*

Exclusiveness states that when a transition is fired in the same execution of a process, the other transitions cannot be fired. An example of exclusiveness is exclusive OR-split, when only one path is followed.

Definition 3.4 (Interleaving Order Relation). *Let $(N,[i])$ be a WF-system. The interleaving order relation $\| \subseteq T \times T$ contains all pairs (x,y) such that $x \succ y$ and $y \succ x$.*

The interleaving order states the absence of any ordering between the occurrences of two activities. An example of this is two transitions being fired in parallel, for example AND-split.

Definition 3.5 (Behavioral Profile (Model)). *For a WF-system $(N,[i])$ the set of behavioral relations $\mathscr{BP} = \{\rightsquigarrow, +, \|\}$ is referred to as the behavioral profile of $(N,[i])$.*

There is an algorithm [30] for finding all the behavioral relations in $O(n^3)$ time, where n is the number of places and transitions in the petri net. The algorithm assumes *sound free-choice* Petri nets.

Behavioral profiles from event logs

It is also possible to compute behavioral profiles purely based on execution logs. One solution for this task is proposed in [32], where behavioral profiles from execution logs are used to measure conformance of the process model. The general idea is very similar to the case of deriving relations from the model. The base for all the relations is still the weak order relation, which in the case of execution logs specifies that given two activities, one happened before the other.

Definition 3.6 (Weak Order (Log)). *Let $L_p = n_1 \dots n_m$ be a log of a process execution and let A_L denote all the different activities present in the log. The weak order relation $\succ_L \subseteq (A_L \times A_L)$ contains all pairs (x,y) such that there exists two indexes $j,k \in \{1,\dots,m-1\}$ with $j < k \leq m$ for which holds $n_j = x$ and $n_k = y$.*

In the case of log files, the weak order specifies which event appears before in the logs.

The strict and interleaving order relations are defined similarly:

Definition 3.7 (Behavioral Profile (Log)). *Let $L_p = n_1 \dots n_m$ be a log of a process model and let A_L denote all the different activities present in the log. A pair $(x, y) \in (A_L \times A_L)$ is in at most one of the following relations*

- *The strict order relation $\rightsquigarrow_L$, iff $x \succ_L y$ and $y \not\succ_L x$.*
- *The interleaving order relation $\|_L$, iff $x \succ_L y$ and $y \succ_L x$.*

The set $\mathcal{BP}_{\mathcal{L}} = \{\rightsquigarrow_{\mathcal{L}}, \|_{\mathcal{L}}\}$ is the behavioral profile for log L.

There are some differences when dealing with behavioral profiles for the log and the model. There is no exclusiveness relation, because we do not observe this based only on a single trace. The exclusiveness relation would occur between all the elements logged and all activities not present in the log, but that might not be correct. An activity might not be executed because of other conditions and therefore we cannot say it was exclusive in relation to some of the activities logged in the log.

We can still consider exclusiveness over all the traces present. Later in this work, we are going to define exclusiveness for log files.

3.2 Business process similarity

When discovering mapping between execution logs and process models, probably the most relevant related work has been done in calculating business process similarity. Finding similarity between process models is crucial when searching for process repositories for similar process models [6, 8]. Also, when alignment [7] between two process models is needed, we need to quantify how well the models match, thus evaluate the similarity. Process alignment is used in merging different models to find similar areas in the model or when displaying visual differences. The similarity becomes core for finding mapping between different processes.

Different modeling notations such as Petri Nets, BPMN, EPC, UML activity diagrams are available and used to denote processes models which provides complications when dealing with similarity. An extra layer of abstraction is used to capture process structure and discard the original modeling language and limitations. Every process can be considered as a directed graph, where nodes denote some activities and directed edges connect nodes. Also, a node labeling is needed to distinguish between different activities.

Definition 3.8 (Business process graph). *Let $\mathcal{L}_{\mathcal{L}}$ be a set of vertex labels and $\mathcal{M}_{\mathcal{L}}$ set of edge labels. A business process graph is a tuple (N, E, λ, μ), in which*

- *N is the set of nodes (activities)*
- *$E \subseteq N \times N$ is the set of edges*

- $\lambda : N \rightarrow \mathscr{L}_{\mathscr{L}}$ *is the function that maps vertices to vertex labels.*
- $\mu : E \rightarrow \mathscr{M}_{\mathscr{L}}$ *is the function that maps edges to edge labels.*

Similar abstractions have been previously used to overcome the usage of different notations [7, 8].

To formalize the similarity problem, we are given two business process graphs $G_1 = (N_1, E_1, \lambda_1, \mu_1)$ and $G_2 = (N_2, E_2, \lambda_2, \mu_2)$. Calculating similarity can now be reduced to calculating the similarity between business process graphs. And when finding alignment between two graphs, we are essentially finding such mapping $M : N_1 \rightarrow N_2$ that maximizes the similarity metric for the models.

There are several ways to approach the similarity problem:

- Node label similarity - using only node labels, we could calculate similarity between each pair of node labels and pair up similar nodes.
- Structural similarity - graph structure holds valuable information about the nodes and how the activities are related to each other. Graph edit distance [14] can be used to calculate similarity.
- Behavioral similarity - using execution semantics of the process models, for example causal footprints [8]. In our work we do not use directly the behavioral similarity, but we incorporate behavioral profiles to represent the execution semantics.

3.2.1 Node similarity

To derive a similarity using node labels, we need to measure differences between two strings. One such method is string edit distance, which states minimal number of atomic operations needed to convert one string to another. The atomic operations are: inserting, deleting and substituting a character. We denote this distance as $sed(s, t)$ where s and t are arbitrary strings. Using the edit distance, we can derive the similarity of two node labels, l_1 and l_2, denoted $sim(l_1, l_2)$ as:

$$sim(l_1, l_2) = 1.0 - \frac{sed(l_1, l_2)}{max(|l_1|, |l_2|)}$$

where $|x|$ denotes the length of string x in characters. We refer to this measure also as syntactic similarity.

For example string edit distance between labels "send order" and "send quote" is five: the word *order* needs to be substituted letter by letter to word *quote*. And the similarity between them is therefore $1.0 - \frac{5}{10} = 0.5$.

This node similarity can already be used to find the mapping. A simple, many-to-many mapping can be derived in the following way. First we need to calculate the similarity between all pairs of labels. The user picks a *cut-off* value so that only these label pairs remain in selection that have larger similarity than the chosen cut off threshold. Remaining pairs compose the many to many mapping, meaning that one label might be possibly mapped to many other labels in the corresponding graph. Although this kind of mapping might not be intuitive, it is the simplest way to derive association between nodes in labelled graphs, given the assumption that associated nodes have similar labels.

Other methods can also be considered to compare node labels, including measures that use synonym information or semantic annotations.

Semantic node matching similarity

String edit distance does not consider semantic similarity between words and therefore may lead to low similarity scores in similar words, that human expert would recognise as similar. For example activity labels *Finish order* and *End order* have syntactic similarity of $sim(\text{finish order}, \text{end order}) = 1.0 - \frac{5}{max(11,10)} = 0.54$, but it is clear that *finish* and *end* are synonyms and ideally, the similarity should be 1.0. The Wordnet database [13] identifies *end* as a synonym for *finish* and therefore it is possible to derive such similarity that uses the synonym information for calculating the score.

This idea has been used to define a semantic similarity metric [8]. Let l_1 and l_2 be two strings, w a function that separates a label into a set of words and $syn(w)$ a function that returns a set of synonyms for a given word w (based on Wordnet dictionary lookup). Let $syn(w_1, w_2)$ be the set of synonyms of w_1 that appear in w_2

$$syn(w_1, w_2) = \cup_{w \in w_1 - w_2} syn(w) \cap (w_2, w_1)$$

Let $w_1 = w(l_1)$ and $w_2 = w(l_2)$ and wi and ws be the weights that associate with identical words and synonymous words.

The semantic similarity is defined as follows

$$sem(l_1, l_2) = \frac{2 \cdot wi \cdot |w_1 \cap w_2| + ws \cdot (|syn(w_1, w_2)| + |syn(w_2, w_1)|)}{|w_1| + |w_2|}$$

Strings are split into words by whitespace and stop words such as “for”,”an” and “a” are removed. Also, all other words are stemmed to their base form using Porter’s stemming algorithm [20].

Possible values for parameters can be $w_i = 1.0$ and $w_s = 0.75$ which were obtained experimentally in work [5].

3.2.2 Structural similarity

Another metric can be defined for the similarity of two business process graphs that uses structural similarity and is based on the graph-edit distance [14]. The graph edit distance between two graphs is the minimal number of graph edit operations that is necessary to transform one graph into another [8]. Edit operations include node deletion or insertion, node substitution and edge deletion or insertion. We can assign each of these operations again a cost, and by counting the operations we can derive the total cost that is needed to convert one graph into another. This cost is the similarity between graphs, as graphs with identical structure require no operations and graphs with a lot of differences need a lot of operations.

For example, consider two graphs G_1 and G_2 that have almost similar structure, only differing by G_2 having one extra node somewhere and also some other node has different label when compared to G_1. There are two operations to transform G_1 into G_2: (1) substitute the node with different label in G_1, (2) add new node to G_1 to match the node int G_2. By these two operations, we have transformed G_1 into G_2.

More formally, to obtain the graph edit distance, we start by first computing a mapping between nodes. The mapping score is found as following:

- For each pair of mapped nodes, we consider them substituted. Their distance is one minus similarity of their labels.
- All nodes that are unmapped are either deleted or inserted.
- An edge is considered to exist only in the other graph if and only the nodes are mapped to nodes in the other graphs and there is an edge between the mapped nodes. Otherwise, the edge is considered deleted or inserted.

The graph edit distance is the weighted average of the fraction of inserted/deleted nodes, the fraction of inserted/deleted edges and the average label similarity of substituted nodes.

Definition 3.9 (Graph edit distance)**.** *Let s* $G_1 = (N_1, E_1, \lambda_1, \mu_1)$ *and* $G_2 = (N_2, E_2, \lambda_2, \mu_2)$ *be two graphs. Let* $M : N_1 \to N_2$ *be a partial injective mapping that matches nodes in* G_1 *with nodes in* G_2*. Lets define domain as* $dom(M) = \{n|(n,m) \in M\}$ *and codomain* $cod(M) = \{m|(n,m) \in M\}$*, and let* $0 \leq wsubn \leq 1$*,* $0 \leq wskipn \leq 1$ *and* $0 \leq wskipe \leq 1$ *be the weights that we assign to the substituted nodes, inserted or deleted nodes and inserted or deleted edges respectively.*

We denote the set of substituted nodes as $subn$, inserted or deleted nodes as $skipn$ and inserted or deleted edges $skipe$ and define them as follows:

$$
\begin{aligned}
subn =& dom(M) \cup cod(M) \\
skipn =& (N_1 \cup N_2) - subn \\
sube =& \{(a,b) \in E_1 | (a,a') \in M, (b,b') \in M, (a',b') \in E_2\} \cup \\
& \{(a,b) \in E_2 | (a,a') \in M, (b,b') \in M, (a',b') \in E_1\} \\
skipe =& (E_1 \cup E_2) - sube
\end{aligned}
$$

The fraction of inserted or deleted nodes, denoted $fskipn$, the fraction of inserted or deleted edges, denoted $fskipe$ and the average distance of substituted nodes, denoted $fsubn$ are defined as follows:

$$
\begin{aligned}
fskipn =& \frac{|skipn|}{|N_1| + |N_2|} \\
fskipe =& \frac{|skipe|}{|E_1| + |E_2|} \\
fsubn =& \frac{2.0 \cdot \sum_{(n,m) \in M} 1.0 - sim(n,m)}{|subn|}
\end{aligned}
$$

The edit distance score of matching is defined as:

$$
\frac{wskipn \cdot fskipn + wskipe \cdot fskipe + wsubn \cdot fsubn}{wskipn + wskipe + wsubn}
$$

The user must still select the appropriate weight values that characterise its expected outcome.

Labelled edges

The graph edit distance does not consider edge labels. Later in our solution we are going to label the edges in the graph with behavioral relations and we want the edit distance to incorporate labelling information.

We consider an edge $e = (n_1, n_2)$ between nodes in graph G_1 matched to an edge $e' = (n_1', n_2')$ in G_2 if and only if the corresponding vertices in G_1 are matched to vertices in G_2 and there exists and edge in G_2 and the labels of the edges are the same, that is $\mu_1(e) = \mu_2(e')$ holds.

The attribute $sube$ in graph edit distance definition (Definition 3.9) becomes as fol-

lows:

$$sube = \{(a,b) \in E_1 | (a,a') \in M, (b,b') \in M, (a',b') \in E_2 \, \mu_1((a,b)) = \mu_2((a',b'))\} \cup$$
$$\{(a,b) \in E_2 | (a,a') \in M, (b,b') \in M, (a',b') \in E_1, \mu_2((a,b)) = \mu_1((a',b'))\}$$

By this modification, we only consider those edges substituted that have th same labels.

3.2.3 Greedy graph matching

Deriving a mapping between two graphs or finding the best edit distance is a NP-complete problem [15]. To find a mapping that produces the smallest edit distance, we are using a greedy algorithm as proposed in [8].

The greedy algorithm (Algorithm 1) works as follows. It starts by calculating all the possible node mappings that have similarity larger than the user supplied $cutoff$, stored in $openpairs$. If $cutoff = 0$ all pairs are generated. In each iteration, the algorithm selects a pair that is open and that increases the score and adds to the mapping. Each node can be added once to the mapping, the algorithms removes all such pairs from $openpairs$ in which one of the selected node appears. The algorithm finishes when there is no open pair left to add to the mapping or none of the open pairs increases the mapping score and therefore no better result is found. The algorithm has time complexity of $O(n^3)$ where n is the number of nodes in the largest graph and quadratic space complexity (the set of open pairs) [6].

Algorithm 1 Greedy algorithm

Input: Two business process graphs $G_1 = (N_1, E_1, \lambda_1, \mu_1)$ and $G_2 = (N_2, E_2, \lambda_2 \mu_2)$, node similarity function sim and mapping scoring function s.

1: $openpairs \Leftarrow \{(n_1, n_2) | n_1 \in N_1, n_2 \in N_2, sim(\lambda_1(n_1), \lambda_2(n_2)) > cutoff\}$
2: $map \Leftarrow \emptyset$
3: **while** exists $(n,m) \in openpairs$, such that $s(map \cup \{(n,m)\}) > s(map)$ and there does not exist another pair $(o,p) \in openpairs$, such that $s(map \cup \{(o,p)\}) > s(map \cup \{(n,m)\})$ **do**
4: $\quad map \Leftarrow map \cup \{(n,m)\}$
5: $\quad openpairs \Leftarrow \{(o,p) \in openpairs | o \neq n, p \neq m\}$
6: **end while**
7: **return** map

One of the drawbacks of the greedy algorithm is that it may result in suboptimal mapping as the algorithm makes choices that seem best at the time. The studies in process alignment and similarity search [9, 6, 7] show that the greedy algorithm provides good enough results when compared some other possibilities, such as A*-star algorithm.

3.3 Similarity flooding algorithm

Similarity flooding [18] is a graph matching technique for finding mapping between any two labelled graphs and using also edge labels. It has been used successfully in the schema matching domain [21]. Here we give a simple overview of the method, for more detailed analysis please refer to the original paper. An illustrative overview of the algorithm is given on Figure 3.1.

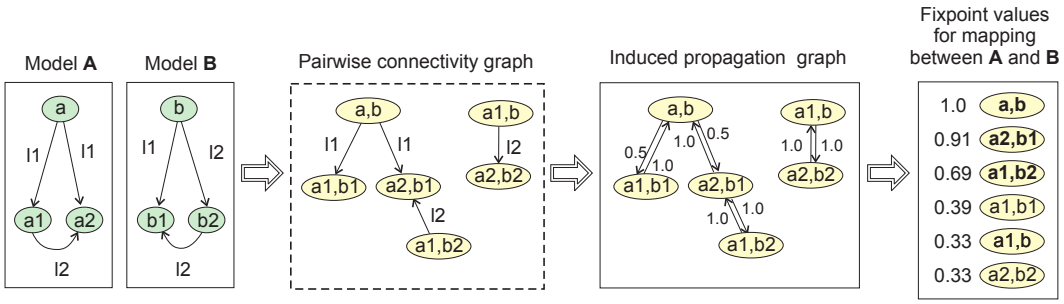


Figure 3.1: Example of the Similarity Flooding algorithm [18].

The algorithm defines the pair-wise connectivity graph (PCG) from the input graphs G_1 and G_2. Each node in the PCG is an element from $N_1 \times N_2$ called *map-pair*. The edges in the connectivity graph are defined as follows:

$$((x_1, y_1), P, (x_2, y_2)) \in PCG(G_1, G_2) \Leftrightarrow (x_1, P, x_2) \in G_1 \text{ and } (y_1, P, y_2) \in G_2$$

Each map-pair contains nodes from both graphs and a similarity score between them, such as semantic similarity. The computation of the algorithm relies on the assumption that a pair of nodes are similar when their adjacent elements are similar. The similarity of two elements is propagated to the PCG to their neighbors as follows:

$$\begin{aligned}\sigma^{k+1}(x,y) = &\sigma^k(x,y) \\ &+ \sum_{(a_i,x)\in G_1,(b_i,y)\in G_2} \sigma^k(a_i,b_i)\cdot W((a_i,b_i),(x,y)) \\ &+ \sum_{(x,a_i)\in G_1,(y,b_i)\in G_2} \sigma^k(a_i,b_i)\cdot W((x,y),(a_i,b_i))\end{aligned}$$

where $\sigma^k(x,y)$ shows the similarity between x and y after iteration k and $W((a_i,b_i),(x,y))$ is the propagation weight of the similarity between a_i and b_i to the similarity between x and y. The similarity propagation is updated iteratively by fixpoint computation and eventually converges. The resulting mapping can be derived from the final iteration scores σ^k, by solving the assignment problem using the Hungarian algorithm [17] to extract the best mapping.

Chapter 4

The mapping discovery method

In this chapter we describe in more detail the mapping between model and logs and outline a general process for finding the mapping. We present the main contribution of this thesis, the method for discovering the mapping between artifact centric process models and process execution logs and discuss its limitations and possible enhancements.

4.1 Problem statement

We are given an artifact-centric process model in Proclet notation and an event log recording the execution of the process model. We are interested in finding for each activity in the log a corresponding activity in the model. Such mapping between activities is needed for conformance checking. In the ideal case, when the log conforms exactly to the model and the activities are named identically in both log and model, extracting such mapping becomes trivial, by associating the corresponding labels.

There are several reasons why extracting the mapping in a realistic setting is not obvious:

- Inconsistency in naming the conventions for activities and events in the log files. This may be caused by multiple reasons, such as there is known mis-conformance between the model and log or even simpler cause, the system implementation did not follow consistent naming convention. Different language constructs may cause difference in naming, for example often activities in the models are named in infinitive case and events in the system logs appear in passive voice or past tense, for example "Send mail" in the model and "Mail sent" entry in the logs.

- Mapping between such model and log pair is needed, where it is known that the system does not follow the supplied model. This happens for example when newer version of the model and older version of the log are used or completely different model is used. In this case it is obvious there is no perfect match and the automatic discovery might reveal a potential mapping.

Disregarding all the differences in the model and logs, we are interested in discovering the mapping automatically. For larger and more complex models, it would be time consuming to deliver the mapping by hand and an automated tool would speed up the process. In general, it is possible always to modify the mapping using input from human experts and the initial output provided can be considered only as baseline.

4.2 Mapping

The mapping consists of multiple parts:

- Mapping from artifact type process models and its activities to events in log files. This association is most important as it is the basis for behavioral conformance checking task.
- Mapping data attributes from event log to data model associated with artifacts. This task is not handled in this thesis, as it is a general form of the schema matching [21] and existing methods such as similarity flooding [18] can be used.

In this thesis we concentrate on activity mapping. The association between events and activities most typically is in the form of one-to-one, where a task is associated with only one log event type and no other task in the log is associated with the same type of log event. As discussed in [22], the mapping must consider following:

- **Duplicated tasks**. Multiple activities in the model may correspond to single event type in the log. In the CD shop example in Figure 2.2 there are two tasks labelled "Add Cd" in *Order* Proclet that are duplicate tasks. By following the traces it is visible which task was executed and the duplicated activities are distinguishable.
- **Invisible tasks**. Some activities are not logged and thus cannot be mapped to the model. This might happen as certain tasks are not monitored by workflow systems, such as procedures requiring human labor, and are impossible to log. Also, empty transitions are used in models for technical reasons, for example to implement conditions and possibility to skip transition firing. In our work, the task cannot be mapped if it is not logged.

Both cases offer challenges in automatic solution delivery if we limit the output to simple, 1-to-1 mapping. Duplicated task should result in the log event mapped to one of the duplicated task.

4.3 Life-cycle of mapping discovery

The process of mapping discovery requires multiple steps (Figure 4.1) that lead from raw event data and Proclet model to a usable mapping. Here we provide short overview and later discuss each step in more detail.

Preprocessing. The first step is to process the log data and extract artifacts from it, as event logs do not explicitly contain artifact information. After preprocessing, we have instance aware logs with user defined or automatically extracted artifact views.

Transformation. To find the mapping, such representation of data and model is needed that enables simple and logical matching and provides reasoning. Proclet model and artifact views on logs are naturally completely different structures and finding associations between these is not logical. A transformation is needed to convert both into a simple, more formal representation. We propose using graphs as a base data structure. To compose the graphs, behavioral profiles are used to incorporate behavioral aspects of both model and logs to the graph structure.

Mapping discovery. Dealing only with graphs, finding the mapping becomes a known problem of schema matching or process alignment.

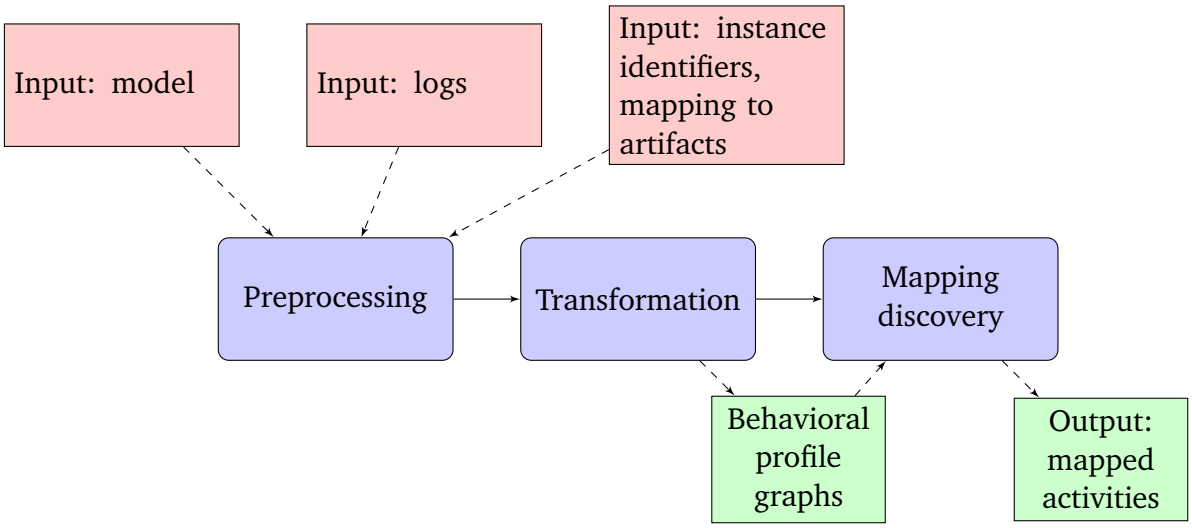


Figure 4.1: Life-cycle of the mapping extraction.

4.4 Preprocessing the data

Artifact-centric systems can store execution info into a relational database or as events in a sequential log similar to classical process logs [10], but without any structuring into cases. Hence, the log contains all activities executed and we can assume that minimally the following data is present in the logs:

```
2007-03-01T13:00:00Z,Accept Quote, quoteId=211,cdno=2203929898
2007-03-01T14:25:00Z,Add quote to Order, quoteId=211, orderid=1232
2007-03-01T17:00:00Z,Sent quote, quoteid=412
2007-04-06T13:00:00Z,Rejected quote, quoteid=1235
```

Figure 4.2: Example of a log with timestamp, event type, and arbitrary list of data attributes.

- Timestamp - time of the event, for keeping ordering of events.
- Event type - the activity executed by the system.
- One or more attributes - data associated with the activity.

A possible example of such log is shown on Figure 4.2.

For an artifact-centric process we need to transform the raw event logs to artifact views, containing execution traces of single artifact instances, in order to use traditional process mining techniques such as conformance checking.

In the logs, the data attributes identify to which artifact instance the event belongs. We discover instance identifiers among the data attributes and group event types that share the same identifier attribute into entities. For example, in the logs in Figure 4.2 we have events with attribute $quoteid$ and we group them into entity $Quote$. From the raw logs as described we discover relations between entities and obtain an entity-relationship model. Entities can be mapped to artifacts in the supplied model by mapping instance identifiers to artifact identifiers.

We can obtain for each artifact the entity mapped to it. We construct traces so that one trace contains events for one instance of the entity. We call this an artifact view on the logs. Due to possible mis conformance, entities and their events might not be exactly the same as the artifacts and their activities.

The whole process of obtaining logs structured into artifact views is beyond the scope of this work and is described in detail in [28].

As a starting point of our work, we can assume that we have obtained logs that are grouped into artifact cases and each corresponds to the execution of one artifact. Considering the CD shop example, we have traces for Order and traces for Quote. Each trace has the case identifier and contains only events associated with this case.

We hereby define some notations that we are going to use

- Instance identifiers as $Inst = (id_1, \ldots, id_n)$ for logs. Instance identifiers specify to which entity event belongs. Based on the example (Figure 4.2) the set of instance identifiers is $Inst = (quoteId, order)$
- Artifacts in the model as $P = (P_1, \ldots, P_m)$

- Mapping between artifacts and the instance identifiers, $M : P \rightarrow Inst$
- Logs for artifact views as $\mathscr{L} = (L_1, \ldots, L_k)$ where each trace L_j contains one execution of a single artifact.

There are two assumptions that we are going to make about the data that will be available when finding the mapping.

- The set of instance identifiers $Inst$ is given. Instance IDs can be supplied by the user or found automatically by preprocessing. This information enables to identify artifacts in the log files.
- Mapping between instance identifiers in the logs and artifacts identifiers in the model is given, that is the mapping $M : P \rightarrow Inst$ is given. With this information we can associate entities in the logs with the corresponding artifacts in the model.

4.5 Transformation

After the preprocessing step, we have logs for each artifact type. Considering the CD shop example, we have obtained separated traces for both *Quote* and *Order* types. On the model side, we have the Proclet model and we are interested in deriving the mapping between event types in the logs and activities in Proclets.

Our idea is to use behavioral profiles to transform logs and model into similar structures. We can extract behavioral profile for each set of logs per artifact type and also for each Proclet in the model. Behavioral relations for both model and log can be represented as a behavioral profile graph (BPG).

For each artifact type specified in the mapping $(p, id) \in M : P \rightarrow Inst$ we extract the graph from the logs, all traces from $\mathscr{L}$ that have corresponding instance identifier id, and the model, the corresponding proclet for P:

Behavioral profile graph for the log. In case of behavioral relation for the logs, the nodes in the graph are all the events found in given logs. If a behavioral relation is present between two events, a directed labeled edge is added in the graph between the corresponding nodes. Edge labelling denotes the type of relation between the nodes. The extraction is done in a straightforward way over each trace and follows the description presented in Section 3.1.

Behavioral profile graph for the model. The corresponding artifact from the Proclet model is extracted by discarding all the ports. The transitions associated to ports lose the connecting arc. For each Proclet, we can in this way obtain the WF-net representing the process model. Using the algorithm described in [29], we extract the behavioral profile for the WF-net and construct the graph similarly as it was done for the log case. Transitions and their labels are the nodes in the BPG and relations between transitions

make up the edge set. Invisible or empty transition in the model are not considered and all relations involving these transitions are discarded.

The obtained graph structures can be compared and mapping can be derived. It must be noted that the graph composed from logs contains only those vertices (event types) and labels that were present in the log, which in general can differ from the set of vertices extracted from the model.

4.5.1 The exclusiveness relation for log case

Original definition for behavioral profiles (Section 3.1) does not contain the exclusiveness relation, since only a single trace is considered. We define the behavioral relation as follows:

Definition 4.1 (Exclusiveness relation (Log)). *Let $\{L_i\}_{i=1}^{n}$ be a set of log traces where $L_i = n_1 \dots n_m$ is a an execution trace of a process model and let A_L denote all the different activities present in the traces. A pair $(x, y) \in (A_L \times A_L)$ is in the exclusiveness relation $+_L$, iff $\forall L_i$, $x \not\prec_{L_i} y$ and $y \not\prec_{L_i} x$.*

We add this relation to the behavioral profile graph for logs.

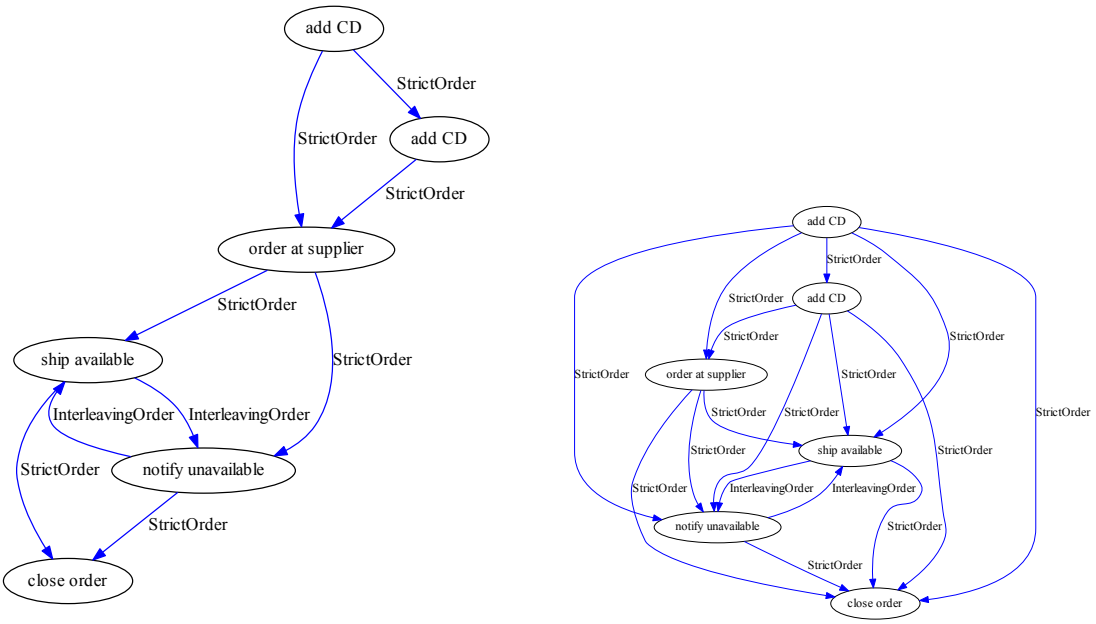
4.5.2 Reducing the set of behavioral relations

In practice, it can be noticed that for different WF-nets, the strict ordering relation is dominating. For example in the CD-shop *Order* model (Figure 2.2) there is a strict ordering relation between *Add cd* and all the other activities in the model. Such excessive strict ordering relations make all BPGs look similar, nodes have similar in-degree and out-degree values and in general the behavioral profiles graphs lack distinct structure. To overcome this, we can filter out strict ordering relations that are not between transitions that are directly preceded one by another.

For WF-nets, we only consider strict ordering relations between transitions, such that there is a connecting place between the transitions under considerations. For the log, we store those relations between event types that always directly precede each other in the log, i.e., no other event appears between them in any execution trace in artifact view format.

The resulting filtering helps to express more the distinctive structure of the model and reduces complexity. As seen on Figure 4.3 the BPG with filtered relations has more resembling structure to the original model (Figure 2.2).

The filtering is based primarily on intuition and it must be noted the filtering may cause inconsistencies between the log and the model, as for the logs, the filtered relations depend on the traces present. But our brief experiments show that these can be overcome for greedy graph edit distance by reducing the weight of the edge matching cost. For



(a) With filtering, number of edges $|E| = 9$.

(b) No filtering, number of edges $|E| = 16$.

Figure 4.3: Behavioral profile graph for *Order* process with (a) and without (b) filtering strict order relations.

Similarity flooding algorithm, the filtering provided somewhat better recall values than without filtering.

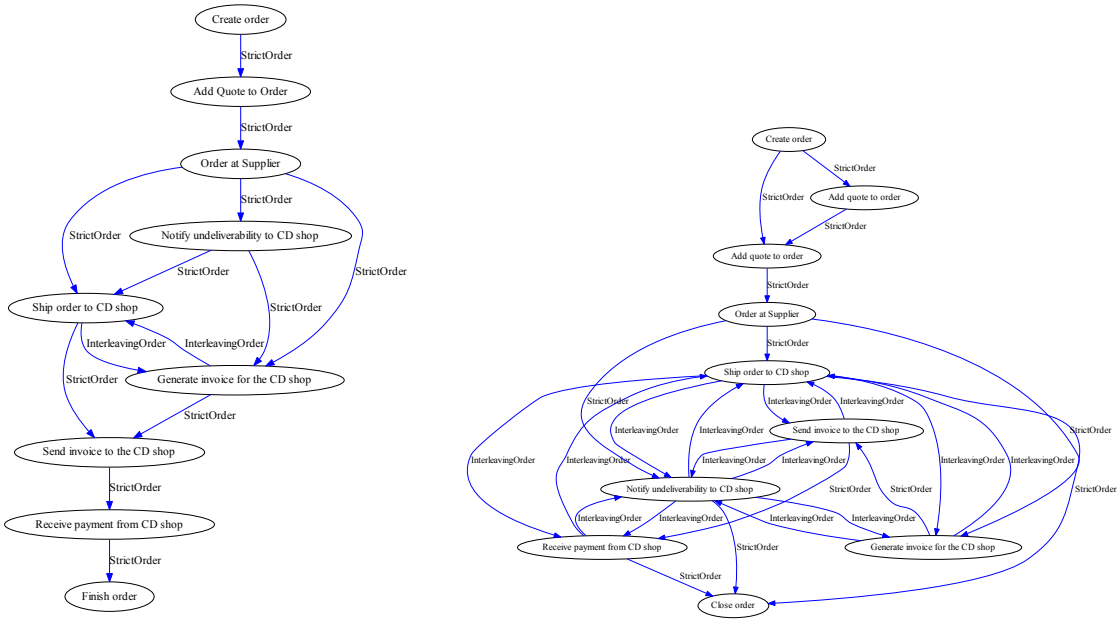
Note that this is not the same as transitively reducing the graph, despite the similarity. The resulting graphs might not be a transitive reduction of the original graphs.

4.6 Mapping discovery

We have now obtained a graph representation for both log files and models and to find the mapping, we only need to consider the BPGs. Example graphs for Order artifact are shown on Figure 4.4.

For each mapped pair between instance identifiers and artifact types specified by the mapping $M : Inst \rightarrow P$, calculate the mapping between the behavioral profile graph from log and the behavioral profile graph of the proclet process model for the artifact type. Mapping is calculated using greedy graph matching algorithm, node similarity is calculated as string edit distance or using the semantic similarity, that recognises synonyms stems the words used.

The mapping produced by the algorithm is partial injective function, meaning that it maps possibly for each event in the log an activity in the model. Each event in log or activity in the model is mapped at most once, but might not be mapped at all. In addition to some events not being mapped at all, the mapping also produces incorrectly mapped



(a) BPG for Order from the log.

(b) BPG for Order from the model.

Figure 4.4: Comparison of behavioral profile graphs from the log(a) and the model(b).

pairs, which may be caused by low similarity of labels or differences in behavioral profile of log and model.

Besides greedy algorithm, there are many other graph matching algorithms to consider [34, 2]. We also experimented with using similarity flooding algorithm [18], which works with directed labelled graphs and is easily adoptable to this case.

4.7 Limitations

One drawback of the proposed method is that it assumes similar organization of activities in the model to artifacts and events in entities, that is, similar structure of artifacts.

With the current proposed solution, an event in the log cannot be assigned to an activity in the model that is located in a different artifact. If the event from the logs is mapped to one entity and the user supplied mapping between entities shows that the event should be ideally in a different artifact, this event is not mapped to the correct activity.

Figure 4.5 shows a different model for the CD-shop system. Our method does not enable to discover the full mapping between the logs and the model supplied. In such scenario, a possible mapping can be extracted by considering each possible pair of entity from the log and artifact from the model. Then we would get multiple, possibly contradicting mappings which should be filtered. For example, Figure 4.5 shows a model with six artifacts and if we have log with two artifacts, we could calculate the possible

mappings between each pair, resulting in twelve mappings.

Despite the limitations, the experiments should give an indication whether the method can produce meaningful mapping. The expected behavior of the method is to map corresponding event types and activities and not to map event types/activities that do not have counterpart in the model/log.

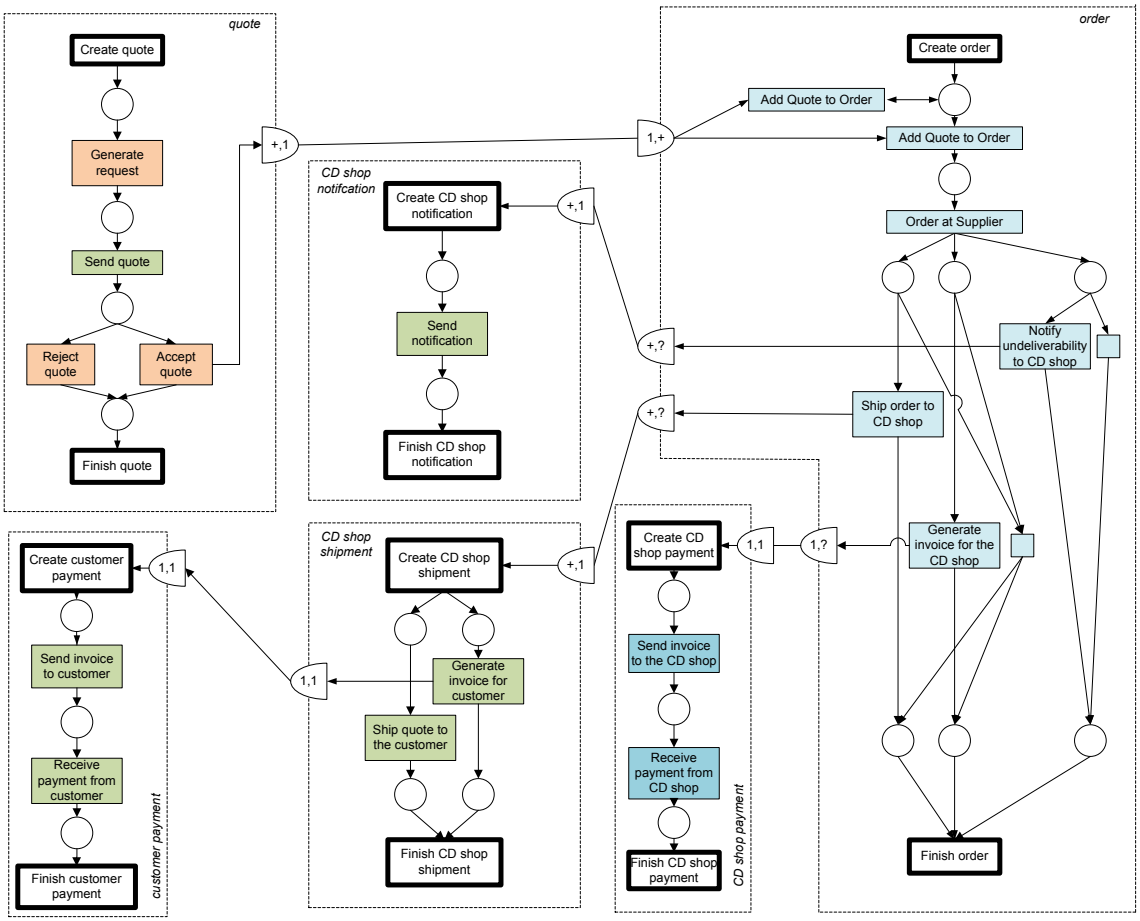


Figure 4.5: The CD shop process model represented as multiple proclets.

4.8 Mapping discovery using combined artifacts

Here we discuss briefly how to overcome the limitation and how the methods should work. This is a general idea and not provided in detail. It is a possible direction for future work.

The idea to solve the problem is to consider all behavioral profiles together in the model. It is more natural to artifact-centric cases, as different artifacts and entities interact and exchange messages.

- The behavioral profiles of the model should be still extracted for each proclet separately. But we can connect the behavioral profile graphs to each other, by con-

```
begin Quote[quoteid = 1] trace
       Generate request[quoteid=1]
       ...
       Create order[orderid=217]
       Add Quote to Order[orderid=217, quoteid=1]
       ...
       Create order[orderid=218]
       ....
       End order[orderid=218]
       ...
       End order[orderid=217]
end quote trace
```

Figure 4.6: Simple interaction between artifacts in a log from Quote viewpoint.

necting the corresponding activities in the BPGs that are connected in the model by ports. We can translate the port to strict ordering relations and by placing these edges, we can get a single BPG for the whole model.

- For logs, we should first extract behavioral profiles for each entity mapped to an artifact. Then derive the ordering information between each pair of related instances of different entities. Instances are related if one refers to the identifier of the other. Figure 4.6 shows a part of a trace for an instance of the Quote entity and the related orders that the quote interacts with. Such trace can be obtained by using related instances and incorporating them into the log. As a result we have a single connected graph for the entities derived from logs.

- Finding matching between these structures. Again, greedy graph matching can be used.

Chapter 5

Experimental evaluation

In this chapter we provide the results of a series of experiments aimed at assessing the quality of mapping produced by the proposed method.

5.1 Overview

To see if the proposed methods can provide a correct output, we carried out a case study on the CD-shop data. The goal is to study whether the correct mapping is discovered between the log and model and how good the mapping is if there are deviations between the logs and model.

The experiments should provide quantitative evaluations on the mapping. Known measures from the information retrieval domain can be used for this purpose. As in the current dataset the actual, i.e., correct mapping is known, we can utilize that information for assessment.

Let $R_A = \{(a_1, b_1), \ldots, (a_i, b_i)\}$ denote the correct mapping as a set of pairs, where pair (x, y) contains activity label x in the model and event name y in the log and i is the number of elements mapped. Similarly, let $R_O = \{(a_1, b_1), \ldots, (a_i, b_i)\}$ be the mapping produced by the algorithm. To assess quality, we are using two measures:

- Precision - the percentage of obtained pairs that are correct. If the resulting output does not contain all pairs, i.e., it does not produce such association that have very low similarity, but produced pairs are correct, the precision would be 1.0. Formally

$$precision = \frac{|R_A \cap R_O|}{|R_O|}$$

- Recall - the percentage of all correct mapped pairs that are present in the result. If our algorithm produces only a single pair of elements that is mapped and it is correct mapping, the precision would be 1.0, but recall $\frac{1}{i}$. There is a trade-off sometimes between precision and recall and we are interested in obtaining high

values in both metrics. Formally

$$recall = \frac{|R_A \cap R_O|}{|R_A|}$$

Note when calculating precision and recall values, we use only a single set of correct mapping over all the artifacts. Also, when there are duplicate tasks in the model, we consider them as single when calculating recall: only one of them can be mapped correctly, if this it so, then in terms of recall, it is correct. If both duplicate tasks are mapped and one them is mapped correctly, we consider on of them as incorrect.

5.2 Implementation details

The mapping discovery is implemented in Java 6. It uses *XES* format as input and hard coded process models as Java Objects to represent Proclet models. The system can read Petri net model for each Proclet separately, for example using XML based *pnml* files.

For extracting behavioral profiles from Petri nets, implementation from open source project JBPT [3] was used. For graph and label matching measures, some of the code originates from [8]. For similarity flooding algorithm, original implementation from [18] was used.

The implementation contains code to read the log files, extract artifacts from the logs and the model, find behavioral relations, build the graphs and finally find the matching.

5.3 Test data and experiments

The CD shop example is used as introduced in Section 2.4.1. A sample scenario of CD-shop description was supplied to several people, who modeled it using Proclets and built Colored Petri net model using CPN-tools software, executed simulation on the model and stored the logs. We have obtained some of the models and logs and study the described method on them.

The logs are preprocessed and are structured into artifact execution cases and their interactions.

We study how the mapping looks on different model and log pairs. Also, we use models and logs in an interchanged configuration. This should simulate real-world conditions, when the mapping is not perfect as some activities inserted/deleted when comparing model and log. The nature of experiments is also to understand what the differences are between different log files.

5.3.1 Process models

Two different models were used in the experiments, that we refer to as *Full model* and *Simple model*.

Full model. The full model (Figure 5.1) is a more complete definition of the CD-shop scenario. It has activities needed to model starting of the quote, adding quote to order, fulfilling the order and shipping it to the shop and to the customer. It also has activities for payment related task, such as *Send invoice to customer*, *Receive payment from CD shop*.

Simple model. We refer to simple model as depicted on Figure 2.2. It has the basic description for the CD-shop scenario, involves two artifacts. Compared to the *full model*, it has less activities and especially lacking activities related to invoicing and payment in *Order* artifact. Compared to the *full model* the labels of activities are more simpler and shorter.

Both models allow multiple quotes to be used per Order and allow a quote to be split up into multiple orders. As they both model the same process, a mapping between activities can be found. Also the interaction between artifacts is the same in both models.

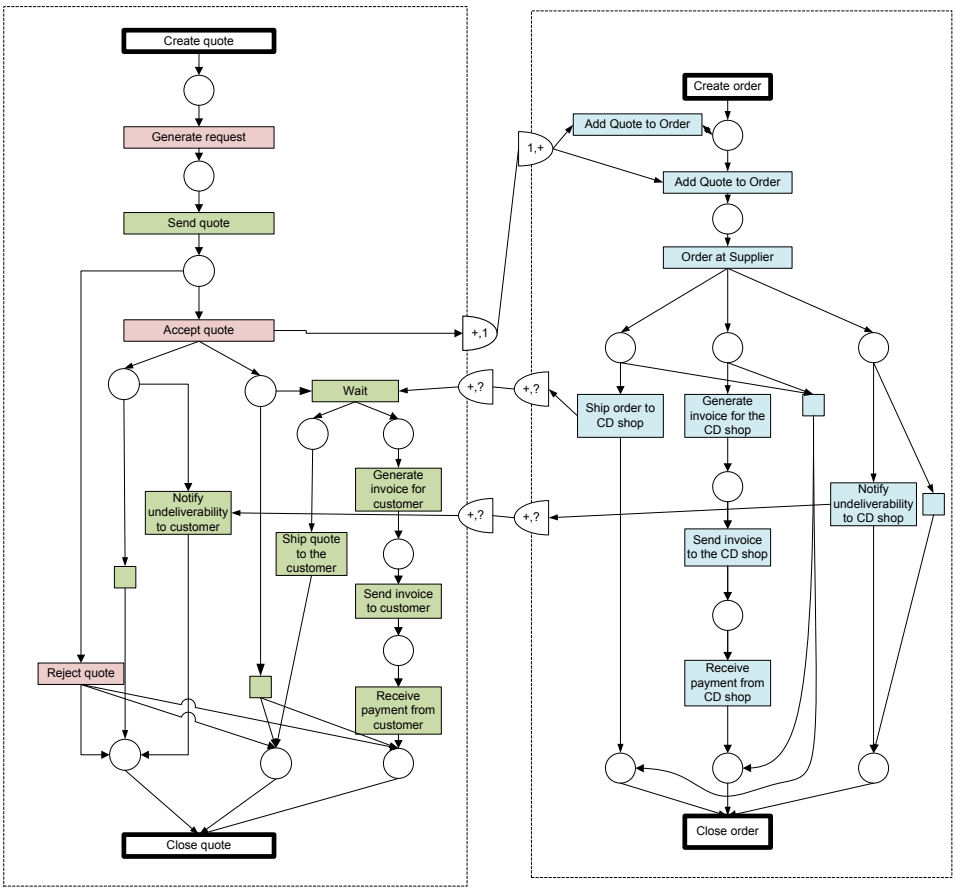


Figure 5.1: The full model.

5.3.2 Log files

Four different log files were used in the experiments.

Simple log. This log corresponds to the simple model, but it is not an exact representation. The log has only seven different events and some activities are missing such as *send quote, reject, deliver, generate invoice* and activities denoting start and end, *close order, close quote, create from request*. Remaining events have the same labels as in the model and interaction between artifacts corresponds to the model.

Full log. Full log corresponds to the full model, with similar model structure and label naming. It only lacks events for activities *Reject quote*, *Create order* and *wait*.

Cyclic log. This log is a variant of the full log, but with a modification that allows reordering of a CD, if it is unavailable. This adds new activity *reorder* to Quote artifact and enables some transitions to be fired multiple times compared to Full log. Also, it has *Reject quote* event present, but lacks invoicing and payment events in the Order artifact.

SQPO log. Single quote per order (SQPO) is variant of full log where an order contains a single CD from a single quote. Multiple orders can be created for a quote. Compared to full log, it has missing events related to close and create of order and quote and also *Add quote to order*.

The logs may have minor differences in label names, such as in full log *Finish quote* and *Close quote* in the model.

5.4 Results

When designing experiments, we were interested in how well the method finds mapping between the log and model pairs that are corresponding and what happens if we mix up different log and model pairs. Also, in the experiments we use both label similarity metrics, string edit distance and semantic distance, and greedy algorithm and similarity flooding algorithm for mapping extraction.

The results of the mapping discovery process are presented as a table in the appendix section. A summary of the results is given in 5.1, which also states precision and recall for each experiment.

If not stated otherwise, all experiments were run using using the greedy graph matching algorithm and string edit distance as node similarity metric. All weights for graph edit distance operations were set to equal: $wskipn = 1.0, wskipe = 1.0, wsubn = 1.0$. Label similarity cutoff threshold was set to 0.2.

5.4.1 Simple log and model

The resulting mapping (Table A.1) between this log and model is perfect. Mostly because of equal labels, the algorithm can easily construct the matching.

5.4.2 Full log and model

In this test, full model and its equivalent log file were used. The resulting mapping can again be considered perfect, as all possible log events are mapped and are mapped correctly. Also, the mapping reveals that *Reject quote, wait, Create quote* are not logged and algorithm does not find a pair for them. Again, most of the labels have similarity of 1.0, except *Finish order* and *Close quote*, that have string edit similarity of 0.58.

5.4.3 SQPO log and full model

The SQPQ log has only single a quote per order and due to this, some activities in the model are not present in the log. Activities *Add quote to order* and *Order at supplier* lose meaning if only a single quote per order is used and are replaced by single *Create order*.

Although some other events are missing in the log (*Close quote, Create quote, wait, close order, Order at Supplier, Add quote to order*), the resulting mapping (Table A.3) is correct and achieves both 1.0 for precision and recall. This is again mostly caused by the fact that the labels have very good similarity.

5.4.4 Cyclic log and full model

The cyclic log introduces a new event in the log *reorder* and also different semantics in logs by enabling reordering in the Quote artifact. Also, it is missing many events from the Order artifact in the logs. The resulting mapping (Table A.4) again is perfect as labels have good similarity.

5.4.5 Simple log and full model

In this experiment, we have now some differences in model and log structure and labels. The labels do not offer perfect similarity of 1.0, so this case should also give information how does the algorithm incorporate graph structure. The resulting mapping (Table A.5) is not good, providing recall of only 0.43. The cause of this result lies in label similarities and weights. The greedy algorithm can not find a pair to extend the mapping that reduces the distances and therefore terminates.

Semantic similarity

When looking at the labels, a human expert would identify similarities, such as pair *(ship available, ship order to CD shop)* is obvious to be suitable for mapping. The string edit distance similarity penalises this similarity by the length and as in this experiment the labels have different length, it seems to be an important factor. We decided to check whether using semantic node similarity would result in a better mapping.

The resulting mapping (Table A.7) when using semantic node similarity removes only one false mapping (*processed, Sent invoice to customer*), but otherwise remains the same.

Weight attribute optimisation

As the semantic similarity did not improve the results, we decided to check if modifying edit distance weight values helps the algorithm to find a better result. To find suitable weight values, we simulated for all the three attributes $wskipn, wskipe, wsubn$ values between $[0..1]$ with a step of 0.1, thus obtaining $10^3 = 1000$ different combinations. From the results, we looked over manually and explored whether there are some rules visible that result in better precision and recall values. One thing we noticed was that if the substitution cost weight $wsubn$ was about twice smaller than other weight attribute values, the resulting mapping had better recall values. Semantic similarity was still used.

One such mapping (Table A.9) misses only one pair (*processed, wait*) and obtains recall of 0.86. The parameter values were $wskipn = 0.7, wsubn = 0.1, wskipe = 0.7$. Reducing the substitution cost can be explained by the fact that as there are no perfect label matches and every new pair added to mapping still affects the edit distance by their similarity and therefore we most penalise the substation cost.

5.4.6 Full log and simple model

With full log and simple model, the result (Table A.6) is similar to the previous experiment, but here even less pairs are mapped, with recall of 0.17 that is, only two pairs out of twelve are found. Compared to the previous setting, this has more events in the log and the corresponding graph. As with the previous experiment, we decided to explore the effects of semantic similarity and different weight values for graph edit operations.

Semantic similarity

Using semantic similarity provided no better results (Table A.8) were obtained as compared to syntactic similarity. The differences in node similarities remain still probably too large to obtain better mapping.

Weight attribute optimisation

By searching better weights we obtained also much better recall and precision values, such as 0.83 for both. The mapping (Table A.10) has five times more mapped pairs than compared to the default weight parameters. The weights used to obtain the result were: $wskipn = 0.2, wsubn = 1.0, wskipe = 0.2$. But here we did not recognise any patterns between the different weight values. What is more interesting, this is somewhat

contrary to the results in Simple log and full model experiment, where we found that low substitution weights yield in better results, but here it is somewhat the contrary.

The false mappings could be probably removed by increasing the label similarity cutoff threshold, as the used value $cutoff = 0.2$ is too wide.

Similarity flooding

We also tested the similarity flooding algorithm. We chose the full log, simple model configuration as it seemed most challenging and did not provide straightforward mapping by good label similarities. The mapping was extracted using Hungarian algorithm and with similarity cutoff of 0.2, that is, those pairs were mapped whose final propagated similarity exceeded 0.2

By using still semantic similarity, the result obtained via Similarity flooding (Table A.11) is worse that the best result obtained by weight vector search, with recall of 0.5. But the similarity flooding is somewhat better here as we do not need to guess any parameters and it just provides the mapping.

We did not go into details with algorithm and provide here just as a reference for comparison, hence no analysis why the result has such score.

5.5 Analysis of results

The experiments show that the method can produce meaningful output and in general the mapping is good. The worst results are produced when the artifact structure does not conform to the log structure, i.e., the cases with mixing different logs and models and the node labels do not match perfectly.

The results (Table 5.1) show that the method tends to always give relatively good precision, i.e., few false mappings but the recall depends on the label similarities. The good precision rate is still present, even though we used very low similarity cutoff score of 0.2.

It can be said that the experiments with mixed settings indicate that node similarity has importance in the result and the mapping depends more on the node labels than probably structure and edge labels. The difference here between syntactic and semantic similarity is minor, but in real-world scenario it may affect the result more. Also, we must note that we did not study the similarity functions and their effects in details and the conclusions are based only on the experiments present..

The possible improvement can rely in better scoring weights in graph edit distance measure. Although, the experiments did not reveal any clear indication which the values should be, in the real-world scenario user can change the values and see how the mapping changes.

Table 5.1: Experiment results.

Experiment	Log File	Model	Precision	Recall
Simple log and model	Simple	Simple	1.0	1.0
Full log and model	Full	Full	1.0	1.0
SQPO log and full model	SQPO	Full	1.0	1.0
Cyclic log and full model	Cyclic	Full	1.0	1.0
Simple log and full model	Simple	Full	0.75	0.43
Full log and simple model	Full	Simple	1.0	0.17
Semantic similarity	Simple	Full	1.0	0.43
Semantic similarity	Full	Simple	1.0	0.17
Weight attribute optimisation	Simple	Full	1.0	0.86
Weight attribute optimisation	Full	Simple	0.83	0.83
Similarity flooding	Full	Simple	0.85	0.5

In addition to greedy algorithm, we used Similarity Flooding algorithm as a comparison. Although the algorithm did not provide perfect results, it still has some advantages as no weight parameters must be chosen. A more detailed analysis would reveal whether this algorithm is suitable for the task.

Chapter 6

Conclusions

In this work we studied the problem of automatically discovering the mapping between activities in artifact-centric process models and process execution logs. Such mapping is needed for conformance checking, where traces are replayed and thus need to know which transition each event represents. Such mapping might not be obvious, as there are possible differences in the structure of model and logs used or in event naming.

The solution expects a model in proclet format and execution log as input. The log must be transformed to artifact views, such that each trace contains the execution of a single entity instance in the log. Our solution for the mapping delivery task considers the artifacts separately in the model. Behavioral profiles are derived for the artifacts in the model and traces and graphs are constructed, denoting activities as nodes and edges indicating behavioral relations between activities. The bevahioral profile graphs are used to transform the entity in the log and the proclet to similar data structures.

The mapping between corresponding activities and events is extracted using the greedy graph matching algorithm, that constructs such matching that minimises the graph edit-distance. For calculating similarity, we experiment with two different metrics, a string edit-distance and semantic similarity, that uses stemming and synonym information.

The method was tested on several different models and logs that all depicted the same scenario. Results show that the method can find the mapping, although qualtit depends on the similarity of the node labels.

One of the limitations of the method is that it expects similar structure for the model and entities in the execution logs. As it considers artifacts separately, it cannot map an event to an activity in different artifact. If an model with different number artifacts is supplied, the method can not provide meaningful mapping.

Future work

The future work related to the problem in this thesis can be extended in many ways.

First, probably the most important part is to extend the method to consider the process model and artifacts in logs both as single graphs. This enables to use models with structures that do not conform to the log and also match events to activities in different artifacts. For this, a method must be devised for extracting communication links between entities from the logs.

Another possible future development should address the problem of automating the whole process and build the method into a stand alone tool or as plug-in for ProM software. The tool should enable user to supply the log and model and extract the mapping, change parameters of the algorithms or enable user to correct some of the mappings.

Finally, a more complex case study should be carried out with larger sample of models and logs and with more complex data, involving more event types and different interactions between artifacts. This would enable to fine tune the graph edit distance parameters and provide proof of the methods ability to work or not.

Artefakti kesksete protsessimudelite ja käivituslogide vaheliste seoste avastamine

Magistritöö (30 EAP)

Riivo Kikas

Resümee

Klassikaliselt on kirjeldatud töövoogusi protsessidele orienteeritud kujul, kus keskendutakse tervele töövoole ja tegevustele selles. Hiljuti on esile kerkinud uudne, artefakti keskne modelleerimine, kus on oluliseks just äriobjektid ning nende vahelised seosed. Artefakti põhised meetodid nõuavad ka muudatusi protsessianalüüsi tehnikates. Üks võimalik protsesside analüüsi meetod on käivituslogide vastavuse kontrollimine protsessi mudeliga, mille abil saab tuvastada kas süsteem käitub nii nagu planeeritud. Mudeli ja logide vastavuse kontrollimiseks on vaja teada, millised sündmused logides vastavad millistele tegevustele mudelis.

Töö eemärgiks on automaatselt tuvastada seosed artefakti põhiste protsessimudelites olevate tegevuste ja töövoosüsteemi logides olevate sündmuste vahel. Selline seose tuvastamine pole triviaalne, kuna võib esineda, et sündmuste nimed logides ja tegevuste nimed mudelis ei ole vastavuses. Näiteks ei jälgita samasid standardeid nimetamisel. Samuti on vaja seoste automaatne tuletamine, kui on teada, et logide ja mudeli vahel on mittesobivused ning kõiki sündmuseid ja tegevusi ei saagi vastavusse viia. Automaatne tuvastamine aitab lihtsustada kasutaja tööd.

Lahenduseks pakutud meetod kasutab sisendina *Procleti* põhist mudelit ja käivituslogi süsteemist. Et leida seos mudeli ja logide vahel, viiakse mõlemad graafi kujule. Seosed leitakse iga artefakti kohta eraldi ning ei kasutata infot nende omavahelise suhtluse kohta. Iga artefakti kohta eraldatakse nende Petri võrk ning koostatakse käitumisrelatsioonid, mis väljendavad kuidas on tegevused antud artefaktis omavahel seotud. Sellest koostatakse graaf, mille tippudeks saavad tegevused ning kaarteks tippude vahel käitumisseosed nende vahel. Analoogselt koostatakse graaf iga logis esinenud olemi kohta. Kasutaja poolt sisestatud olemite ja artefaktide tüüpide vahelise seoste abil leitakse iga vastava olemi ja artefakti isendi tegevuste ja sündmuste vahelised seosed. Seoste leidmine taandub kahe graafi vaheliste tippude kujutuse leidmisele. Seoste leidmiseks esmalt arvutatakse sarnasused tegevuste nimede vahel ning selle põhjal leitakse kujutus, mis minimiseeriks teisenduskaugust graafide vahel antud kujutuse põhjal. Kujutuse leimiseks kasutatakse ahnet algoritmi.

Praktilise eksperimendina testiti meetodit erinevate mudelite ja logide kombinatsioonidel. Tulemused näitavad, et meetod on võimeline seoseid leidma, kuid tulemuste kvaliteet sõltub palju tegevuste ja sündmuste nimede sarnasusest ja vähem struktuurilisest sarnasustest.

Appendix A

Experiment outputs

Each table here lists the output of the mapping for specific experiment. A table is divided into two sections, one per artifact. The columns *Log* and *Model* and their contents specify the mapping obtained by the algorithm. If the third column *Correct* has "+",the mapped pair is correct, "-" for incorrect. If an activity was not mapped, its counterpart column and the *Correct* column are left empty.

Table A.1: Simple log and model

Log	Model	Correct
Artifact Quote		
processed	processed	+
notify unavailability	notify unavailability	+
accept	accept	+
	send quote	
	reject	
	deliver	
	close quote	
	generate invoice	
	create from request	
Artifact Order		
Notify unavailable	notify unavailable	+
ship available	ship available	+
add CD	add CD	+
order at supplier	order at supplier	+
	add CD	
	close order	

Table A.2: Full log and model

Log	Model	Correct
Artifact Quote		
Send quote	Send quote	+
Receive payment from customer	Receive payment from customer	+
Sent invoice to customer	Sent invoice to customer	+
Notify undeliverability to customer	Notify undeliverability to customer	+
Generate invoice for customer	Generate invoice for customer	+
Generate request	Generate request	+
Ship quote to the customer	Ship quote to the customer	+
Finish quote	Close quote	+
Accept quote	Accept quote	+
	Reject quote	
	wait	
	Create quote	
Artifact Order		
Generate invoice for the CD shop	Generate invoice for the CD shop	+
Add Quote to Order	Add quote to order	+
Order at Supplier	Order at Supplier	+
Notify undeliverability to CD shop	Notify undeliverability to CD shop	+
Create order	Create order	+
Receive payment from CD shop	Receive payment from CD shop	+
Finish order	Close order	+
Send invoice to the CD shop	Send invoice to the CD shop	+
Ship order to CD shop	Ship order to CD shop	+
	Add quote to order	

Table A.3: SQPO log and full model

Log	Model	Correct
Artifact Quote		
Receive payment from customer	Receive payment from customer	+
Ship to the customer	Ship quote to the customer	+
Send quote	Send quote	+
Sent invoice to customer	Sent invoice to customer	+
Notify undeliverability to customer	Notify undeliverability to customer	+
Generate request	Generate request	+
Generate invoice for customer	Generate invoice for customer	+
Reject quote	Reject quote	+
Accept quote	Accept quote	+
	Close quote	
	wait	
	Create quote	
Artifact Order		
Notify undeliverability to CD shop	Notify undeliverability to CD shop	+
Ship order to CD shop	Ship order to CD shop	+
Create order	Create order	+
Send invoice to the CD shop	Send invoice to the CD shop	+
Receive payment from CD shop	Receive payment from CD shop	+
Generate invoice for the CD shop	Generate invoice for the CD shop	+
	Close order	
	Order at Supplier	
	Add quote to order	
	Add quote to order	

Table A.4: Cyclic log and full model

Log	Model	Correct
Artifact Quote		
Ship to customer	Ship quote to the customer	+
Receive payment from customer	Receive payment from customer	+
Notify undeliverability_quote	Notify undeliverability to customer	+
Send quote	Send quote	+
Generate request	Generate request	+
Generate invoice for customer	Generate invoice for customer	+
Reject quote	Reject quote	+
Accept quote	Accept quote	+
Close quote	Close quote	+
Send invoice to customer	Sent invoice to customer	+
reorder		
	wait	
	Create quote	
Artifact Order		
add quote to order	Add quote to order	+
Ship order to CD shop	Ship order to CD shop	+
Notify undeliverability_order	Notify undeliverability to CD shop	+
	Close order	
	Send invoice to the CD shop	
	Receive payment from CD shop	
	Generate invoice for the CD shop	
	Order at Supplier	
	Create order	
	Add quote to order	

Table A.5: Simple log and full model

Log	Model	Correct
Artifact Quote		
processed	Sent invoice to customer	-
notify unavailability	Notify undeliverability to customer	+
accept	Accept quote	+
	Generate request	
	Generate invoice for customer	
	Reject quote	
	Close quote	
	wait	
	Receive payment from customer	
	Ship quote to the customer	
	Create quote	
	Send quote	
Artifact Order		
order at supplier	Order at Supplier	+
Notify unavailable		
ship available		
add CD		
	Close order	
	Receive payment from CD shop	
	Ship order to CD shop	
	Notify undeliverability to CD shop	
	Send invoice to the CD shop	
	Generate invoice for the CD shop	
	Add quote to order	
	Create order	
	Add quote to order	

Table A.6: Full log and simple model

Log	Model	Correct
Artifact Quote		
Send quote	send quote	+
Generate request		
Notify undeliverability to customer		
Finish quote		
Generate invoice for customer		
Accept quote		
Ship quote to the customer		
Sent invoice to customer		
Receive payment from customer		
	processed	
	reject	
	notify unavailability	
	deliver	
	accept	
	close quote	
	create from request	
	generate invoice	
Artifact Order		
Order at Supplier	order at supplier	+
Add Quote to Order		
Finish order		
Ship order to CD shop		
Send invoice to the CD shop		
Receive payment from CD shop		
Notify undeliverability to CD shop		
Generate invoice for the CD shop		
Create order		
	ship available	
	notify unavailable	
	add CD	
	add CD	
	close order	

Table A.7: Simple log and full model, semantic similarity

Log	Model	Correct
Artifact Quote		
notify unavailability	Notify undeliverability to customer	+
accept	Accept quote	+
processed		
	Generate request	
	Generate invoice for customer	
	Reject quote	
	Close quote	
	wait	
	Sent invoice to customer	
	Receive payment from customer	
	Ship quote to the customer	
	Create quote	
	Send quote	
Artifact Order		
order at supplier	Order at Supplier	+
Notify unavailable		
ship available		
add CD		
	Close order	
	Ship order to CD shop	
	Send invoice to the CD shop	
	Notify undeliverability to CD shop	
	Receive payment from CD shop	
	Generate invoice for the CD shop	
	Add quote to order	
	Create order	
	Add quote to order	

Table A.8: Full log and simple model, semantic similarity

Log	Model	Correct
Artifact Quote		
Send quote	send quote	+
Generate request		
Notify undeliverability to customer		
Finish quote		
Generate invoice for customer		
Accept quote		
Ship quote to the customer		
Sent invoice to customer		
Receive payment from customer		
	processed	
	reject	
	notify unavailability	
	deliver	
	accept	
	close quote	
	create from request	
	generate invoice	
Artifact Order		
Order at Supplier	order at supplier	+
Add Quote to Order		
Finish order		
Ship order to CD shop		
Send invoice to the CD shop		
Receive payment from CD shop		
Notify undeliverability to CD shop		
Generate invoice for the CD shop		
Create order		
	ship available	
	notify unavailable	
	add CD	
	add CD	
	close order	

Table A.9: Simple log and full model, semantic similarity, weights: $wskipn = 0.7, wsubn = 0.1, wskipe = 0.7$

Log	Model	Correct
Artifact Quote		
notify unavailability	Notify undeliverability to customer	+
accept	Accept quote	+
processed		
	Generate request	
	Generate invoice for customer	
	Reject quote	
	Close quote	
	wait	
	Receive payment from customer	
	Ship quote to the customer	
	Sent invoice to customer	
	Create quote	
	Send quote	
Artifact Order		
ship available	Ship order to CD shop	+
Notify unavailable	Notify undeliverability to CD shop	+
add CD	Add quote to order	+
order at supplier	Order at Supplier	+
	Close order	
	Receive payment from CD shop	
	Send invoice to the CD shop	
	Generate invoice for the CD shop	
	Create order	
	Add quote to order	

Table A.10: Full log and simple model, semantic similarity, weights: $wskipn = 0.2, wsubn = 1.0, wskipe = 0.2$

Log	Model	Correct
Artifact Quote		
Generate request	create from request	+
Notify undeliverability to customer	notify unavailability	+
Send quote	send quote	+
Generate invoice for customer	generate invoice	+
Ship quote to the customer	close quote	-
Accept quote	accept	+
Finish quote		
Sent invoice to customer		
Receive payment from customer		
	processed	
	reject	
	deliver	
Artifact Order		
Notify undeliverability to CD shop	notify unavailable	+
Ship order to CD shop	ship available	+
Order at Supplier	order at supplier	+
Receive payment from CD shop	add CD	-
Finish order	close order	+
Add Quote to Order	add CD	+
Send invoice to the CD shop		
Generate invoice for the CD shop		
Create order		

Table A.11: Full log and simple model, semantic similarity, similarity flooding algorithm, mapping derived using Hungarian algorithm and only with pairs of final similarity larger 0.2 included in the mapping

Log	Model	Correct
Artifact		
Generate request	create from request	+
Send quote	send quote	+
Ship quote to the customer	deliver	+
Generate invoice for customer	generate invoice	+
Notify undeliverability to customer		
Finish quote		
Accept quote		
Sent invoice to customer		
Receive payment from customer		
	processed	
	reject	
	notify unavailability	
	accept	
	close quote	
Artifact		
Receive payment from CD shop	ship available	-
Order at Supplier	order at supplier	+
Add Quote to Order	add CD	+
Finish order		
Ship order to CD shop		
Send invoice to the CD shop		
Notify undeliverability to CD shop		
Generate invoice for the CD shop		
Create order		
	notify unavailable	
	add CD	
	close order	

Bibliography

[1] Wil M. P. Van Der Aalst. Business process management demystified: A tutorial on models, systems and standards for workflow management. *Lectures on Concurrency and Petri Nets*, pages 21–58, 2004.

[2] Horst Bunke. Graph matching: Theoretical foundations, algorithms, and applications. In *International Conference on Vision Interface*, pages 82–88, 2000.

[3] Business Process Technologies 4 Java. http://code.google.com/p/jbpt/.

[4] David Cohn and Richard Hull. Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. *IEEE Data(base) Engineering Bulletin*, 32:3–9, 2009.

[5] Remco Dijkmam, Marlon Dumas, Boudewijn van Dongen, Reina Käärik, and Mendling Jan. Measuring similarity between business process models. *Information Systems*, Volume 36(2):498–516, 2011.

[6] Remco Dijkman, Marlon Dumas, and Luciano. Garcia-Bañuelos. *Graph matching algorithms for business process model similarity search*, volume 5701, pages 48–63. Springer Berlin / Heidelberg, 2009.

[7] Remco Dijkman, Marlon Dumas, Luciano Garcia-Banuelos, and Reina Kaarik. Aligning Business Process Models. *2009 IEEE International Enterprise Distributed Object Computing Conference*, pages 45–53, September 2009.

[8] Remco Dijkman, Marlon Dumas, Boudewijn van Dongen, Reina Käärik, and Jan Mendling. Similarity of business process models: Metrics and evaluation. *Information Systems*, 36(2):498–516, April 2011.

[9] Marlon Dumas, Luciano Garcia-Bañuelos, and Remco Dijkman. Similarity search of business process models. *IEEE Data Eng. Bull*, 32(3):23–28, 2009.

[10] Dirk Fahland, M. de Leoni, B.F. van Dongen, and W.M.P. van der Aalst. Checking Behavioral Conformance of Artifacts. Technical report, http://bpmcenter.org/wp-content/uploads/reports/2011/BPM-11-08.pdf (Accessed 10.05.2011), 2011.

[11] Dirk Fahland, Massimiliano De Leoni, Boudewijn F Van Dongen, and Wil M. P. Van Der Aalst. Behavioral Conformance of Artifact-Centric Process Models. *Proceedings of the 14th International Conference on Business Information Systems,BIS 2011,Poznan, Poland, June 15–17, 2011.*, 2011.

[12] Dirk Fahland, Massimiliano De Leoni, Boudewijn F Van Dongen, and Wil M P Van Der Aalst. Many-to-Many : Some Observations on Interactions in Artifact Choreographies. In *Service under ihre Komposition (Proceedings of the 3rd Central-European Workshop, ZEUS 2011, Karlsruhe, Germany, February 21-22, 2011)*, pages 9–15, 2011.

[13] Christiane Fellbaum. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, 1998.

[14] X Gao, B Xiao, D Tao, and X Li. A survey of graph edit distance. *Pattern Analysis and Applications*, 13(1):113–129, 2010.

[15] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co. New York, NY, USA, November 1990.

[16] K Jensen. Coloured Petri Nets. *Petri nets central models and their properties*, 254:248–299, 1987.

[17] Harold W Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[18] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: a versatile graph matching algorithm and its application to schema matching. *Proceedings 18th International Conference on Data Engineering*, 2002:117–128, 2002.

[19] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, February 1981.

[20] M F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[21] Erhard Rahm and Philip a. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, December 2001.

[22] Anne Rozinat and WMP Van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, March 2008.

[23] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[24] Wil M. P. van der Aalst, P Barthelmess, C. Ellis, and J Wainer. Workflow modeling using proclets. In *Cooperative Information Systems*, pages 198–209. Springer, 2000.

[25] W.M.P. Van der Aalst. The application of Petri nets to workflow management. *Journal of Circuits Systems and Computers*, 8:21–66, 1998.

[26] W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, AK Alves de Medeiros, M. Song, and HMW Verbeek. Business process mining: An industrial application. *Information Systems*, 32(5):713–732, 2007.

[27] B.F. van Dongen, A.K.A. de Medeiros, HMW Verbeek, A. Weijters, and Wil M. P. Van Der Aalst. The ProM framework: A new era in process mining tool support. *Applications and Theory of Petri Nets 2005*, pages 444–454, 2005.

[28] Boudewijn van Dongen, Marlon Dumas, Dirk Fahland, Massimiliano de Leoni, and Viara Popova. D3.1 Conformance Checker, Deliverable, ACSI. Technical report, http://www.acsi-project.eu/deliverables/D3.1_Conformance_Checker.pdf, Checked 10.06.2011, 2011.

[29] Matthias Weidlich, Felix Elliger, and Mathias Weske. Generalised Computation of Behavioural Profiles based on Petri-Net Unfoldings. In *WS-FM'10 Proceedings of the 7th international conference on Web services and formal methods*, 2011.

[30] Matthias Weidlich, Jan Mendling, and Mathias Weske. Computation of Behavioural Profiles of Processes Models. Technical report, Technical report, BPT Technical Report 07, 2009, 2009.

[31] Matthias Weidlich, Jan Mendling, and Mathias Weske. Efficient consistency measurement based on behavioural profiles of process models. *IEEE Transactions on Software Engineering*, pages 1–21, 2010.

[32] Matthias Weidlich, Artem Polyvyanyy, Nirmit Desai, and Jan Mendling. Process compliance measurement based on behavioural profiles. In *Advanced Information Systems Engineering*, pages 499–514. Springer, 2010.

[33] A. Weijters and WMP Van der Aalst. Process mining: discovering workflow models from event-based data. In *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001)*, number i, pages 283–290. Citeseer, 2001.

[34] L Zager and G Verghese. Graph similarity scoring and matching. *Applied Mathematics Letters*, 21(1):86–94, 2008.