Requirements for process, resource and compliance rules extraction from text

Deliverable D3.1

FFG – IKT der Zukunft
SHAPE Project
2014 – 845638

Table 1 Document Information

Project acronym:	SHAPE
Project full title:	Safety-critical Human- & dAta-centric Process management in Engineering projects
Work package:	3
Document number:	3.1
Document title:	Requirements for process, resource and compliance rules extraction from text
Version:	1
Delivery date:	01 April 2015 (M1)
Actual publication date:	————
Dissemination level:	Public
Nature:	Report
Editor(s) / lead beneficiary:	WU Vienna
Author(s):	Saimir Bala, Cristina Cabanillas, Jan Mendling, Axel Polleres
Reviewer(s):	Claudio Di Ciccio, Alois Haselboeck

Contents

1 Introduction

Text mining is a broad field of study that allows for the extraction of information for the generation of knowledge from natural language text. Process mining is a relatively newer field, even though rapidly growing, that aims at extracting process descriptions from past process executions stored in event logs. The goal of work package 3 (WP3) is to combine methods from these two fields to derive rich process models by analyzing information from structured and unstructured data sources. The outcoming process models can then be used for compliance checking purposes, which constitute the primary goals of the SHAPE project.

In this view, the content of this deliverable is structured as follows: Section 2 presents a literature overview on text mining introducing the main vocabulary as well as potential applicabilities of this research field in SHAPE. Section 3 focuses on process mining and provides insights of the three types of process mining along with a new approach for mining the organizational perspective of business processes, which is one of the issues to be addressed in SHAPE, as seen in the list of requirements from the domain presented in Deliverable 4.1 Cabanillas et al. [2015a]. Section 4 introduces a new method for discovering *project-oriented business processes*, a class of business processes that are executed only once with well-defined goals, budget, resources, and time. Finally, Section 5 concludes the deliverable.

2 Text Mining

This section provides a literature overview on text mining, followed by considerations on how text mining can be used to address the goals of WP3.

2.1 Literature review

Text mining refers to the process of deriving information from natural language text. It relates to data mining in that both strive to extract meaningful information from raw data. However, data mining is characterized as the extraction of implicit, previously unknown, and potentially useful information from data, whereas with text mining the information to be extracted is clearly and explicitly stated in the text [Witten et al., 1999].

Text mining can be regarded as going beyond information access to further help users analyze and digest information and facilitate decision making. There are also many applications of text mining where the primary goal is to analyze

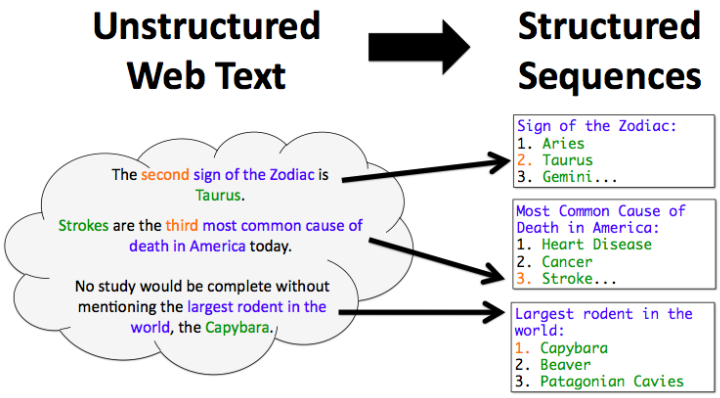


Figure 1 Extraction of structured information from unstructured sources

and discover any interesting patterns, including trends and outliers, in text data, and the notion of a query is not essential or even relevant.

Although text mining is mostly about Natural Language Processing (NLP) [Jurafsky and Martin, 2014], it embraces also applications that go beyond, such as analysis of linkage structures such as citations in the academic literature and hyperlinks in the Web literature, both useful sources of information that lie outside the traditional domain of natural language processing.

This literature review aims to bring an overview on the broad field of text mining with the purpose of covering the main aspects. It is further organized in subsections that consider the different applications of text mining.

2.1.1 Information extraction

Information extraction is used to refer to the task of filling in templates from natural language text. The goal is to extract from the documents (which may be in a variety of languages) salient facts about prespecified types of events, entities or relationships. These facts are then usually entered automatically into a database, which may then be used to analyze the data for trends, to give a natural language summary, or simply to serve for on-line access.

Traditional information extraction techniques (Cowie and Lehnert [1996], Mooney [1999]) leverage on rule-based systems that match predefined linguistic patterns. More recently, work on named entity recognition uses statistical machine learning methods (Seymore et al. [1999]). A tool that uses unsupervised learning can be found in Banko et al. [2007].

2.1.2 Topic detection and tracking

Topic Detection and Tracking (TDT) was a DARPA-sponsored initiative to investigate on finding and following new events in a stream of broadcast news stories. The TDT problem consists of three major tasks: (1) segmenting a stream of data, especially recognized speech, into distinct stories; (2) identifying those news stories that are the first to discuss a new event occurring in the news; and (3) given a small number of sample news stories about an event, finding all following stories in the stream. The work of Allan et al. [1998] has formally defined this problem and proposed the initial set of algorithms for the task. Main subtasks of TDT, as identified in Wayne [2000] are (i) finding topically homogeneous regions (segmentation); (ii) finding additional stories about a given topic (tracking); (iii) detecting and threading together new topics (detection); (iv) Detecting new topics (first story detection); and (v) Deciding whether stories are on the same topic (linking). An example of a real-world TDT system is Google Alerts[1].

2.1.3 Summarization

Summarization is the task of reducing the content obtained from text documents, still keeping a brief overview on a topic that they treat. Summarization techniques generally fall into two categories (Aggarwal and Zhai [2012]). In extractive summarization, a summary consists of information units extracted from the original text; in contrast, in abstractive summarization, a summary may contain "synthesized" information units that may not necessarily occur in the text document. An automatic summarization process can be divided into three steps (Gupta and Lehal [2009]): (1) In the preprocessing step a structured representation of the original text is obtained; (2) In the processing step an algorithm must transform the text structure into a summary structure; and (3) In the generation step the final summary is obtained from the summary structure. A plethora of text summarization tools can be found online. A few examples are the open source libraries such Open Text Summarizer[2] and MEAD[3], as well as the free online tools Sumplify[4] or Online summarize tool[5].

[1] https://www.google.com/alerts
[2] http://libots.sourceforge.net/
[3] http://www.summarization.com/mead/
[4] http://sumplify.com/
[5] http://www.tools4noobs.com/summarize/

2.1.4 Categorization

Categorization involves identifying the main themes of a document. This translates into assigning natural language documents to predefined categories according to their content (Sebastiani [2002]). Categorization often relies on a thesaurus for which topics are predefined, and relationships are identified by looking for broad terms, narrower terms, synonyms, and related terms. Support Vector Machines (SVM)s are used to automatically learn text classifiers from examples (Joachims [1998]). Categorization tools usually rank documents according to how much of their content fits in a particular topic.

2.1.5 Clustering

Clustering is a technique used to group similar documents, but it differs from categorization in that documents are clustered on the fly instead of through predefined topics. Documents can also appear in multiple subtopics, ensuring that useful documents are not omitted from the search results. A basic clustering algorithm creates a vector of topics for each document and measures the weights of how the document fits into each cluster. A survey of clustering algorithms can be found in [Aggarwal and Zhai, 2012, Chap. 4]

2.1.6 Concept Linkage

Concept-linkage tools connect related documents by identifying their shared concepts, helping users find information they perhaps wouldn't have found through traditional search methods (Gupta and Lehal [2009]). It promotes browsing for information rather than searching for it. For example, a text mining software solution may easily identify a transitive closure in a set of topics {X, Y, Z}, i.e., a link between X and Y, a link between Y and Z, and a link between X and Z. With large sets of data, such a relationship could be disregarded by humans.

2.1.7 Information visualization

Information visualization aims at visualizing large textual sources in such a way that the content can be displayed in a hierarchy or map and provides browsing features, in addition to simple search. For instance, governments or police can identify terrorist networks in a map and identify crimes that were previously unconnected (Gupta and Lehal [2009]).

2.1.8 Question answering

Another application area of developed text-mining technologies, along with the natural language processing, is natural language features Question Answering (QA). QA is concerned with building systems that automatically answer questions posed by humans in a natural language. One of the first Query Answering tools has been START[6], developed in the work of Katz [1997]. Question answering has been extensively used in Biomedical domain for aiding researchers and health care professionals in managing the continuous growth of information.

2.1.9 Association rule mining

The focus of association rules mining is to study the relationships and implications among topics, or descriptive concepts, that are used to characterize a corpus. The work of **?** shows how it is possible to discover association rules from massive data from databases, referred to as *basket* data. The same approach can be followed by constructing a database of rules using information extraction methods and subsequently applying techniques (e.g. **?**) to uncover hidden associations in the database. For instance, a rule might be that 98% of customers that purchase tires and auto accessories also get automotive services done. This suggests that association rules can be captured as if/then patterns. Criteria such as support and confidence (Agrawal et al. [1993]) are used to identify the most important relationships. Support is an indication of how frequently the items appear in the database. Confidence indicates the number of times the if/then statements has been found to be true.

2.2 Text mining in SHAPE

The goal of WP3 is to create structured information from unstructured data. In the railway domain, processes are often described in natural language in free-formatted documents, or implicitly in the communication channels between project participants (customers, engineers, managers), such as emails exchanged during process execution (**?**). Electronic logs of task executions, known as event logs, also store information about how processes were executed in the past. All that data shall be transformed and organized in structured process models.

The aforementioned information is available in SHAPE in the form of handbooks and logbooks that were manually compiled by railway engineers. Further

[6] http://start.csail.mit.edu/index.php

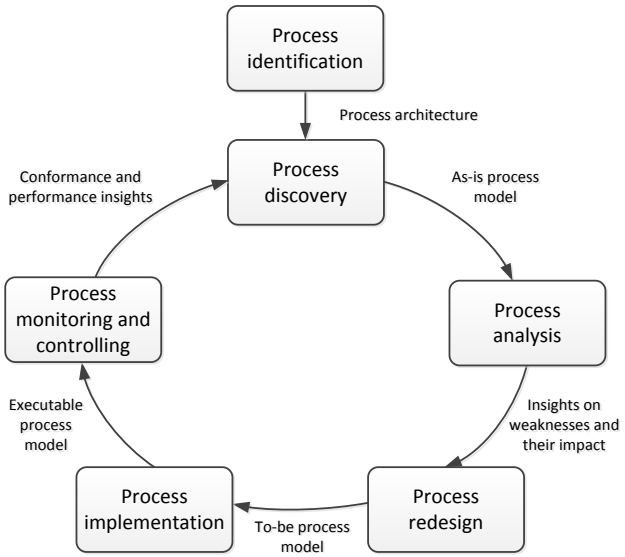


Figure 2 The BPM life cycle as shown in [Dumas et al., 2013]

data are Version Control System (VCS) logs that come from several tools and prior projects. These documents may generally reflect the processes that were followed by the engineers.

Text mining techniques can be used to gather entities and relations from the handbooks. Furthermore, NLP (Jurafsky and Martin [2014]) techniques can be used for extracting, parsing and removing ambiguity on the text. It is then possible to use semantic annotations to unambiguously associate meanings to words that relate to activities of the processes. In this way, the process workflow can be reconstructed based on the temporal and semantic relations that can be found on the text.

3 Process Mining from Event Logs

Before describing process mining, let us introduce the *BPM life cycle* (cf. Figure 2) in order to see where process mining is positioned.

The BPM life cycle describes the different phases of managing a particular business process. The life cycle begins with the process identification, where processes are defined for the first time. In the process discovery phase, existing processes are discovered that may be relevant for pursuing defined objectives. The output of this phase is a process model. Process analysis deals with the goodness of processes both from qualitative and quantitative perspectives. Process implementation transforms a processes model into an executable model that can be run in a Business Process Management System (BPMS). Process monitoring and control-

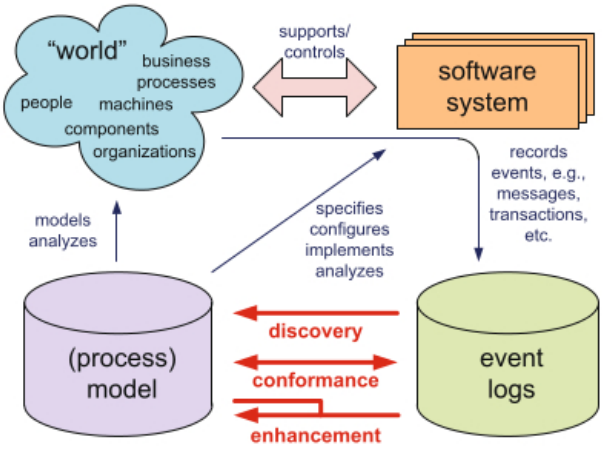


Figure 3 The process mining framework

ling is the phase in which a process is already running. The process still needs to be monitored and supervised for possible changes. When a change is required, a new iteration or the BPM life cycle starts over again.

Only severe problems or major external changes will trigger another iteration of the life cycle, and factual information about the current process is not actively used in redesign decisions. Process mining offers the possibility to truly "close" the BPM life cycle (van der Aalst [2011a]). In fact, data stored in information systems can be used to give an clearer overview on the actual process, in such a way that it can be easily monitored.

Process mining is an affirmed research field that lies between machine learning and data mining on one side and process modeling and analysis on the other side. The goal of process mining is to use event data to extract process-related information, e.g., to automatically discover a process model by observing events recorded by some enterprise system.

Figure 3 illustrates the process mining framework. Relying on the event logs, process mining allows for three main tasks: (i) discovery; (ii) conformance checking; and (iii) process enhancement.

Process mining techniques can also be used in an online setting. We refer to this as operational support. An example is the detection of non-conformance at the moment the deviation actually takes place. Another example is time prediction for running cases, i.e., given a partially executed case the remaining processing time is estimated based on historic information of similar cases. This illustrates that the "process mining spectrum" is broad and not limited to process discovery. In fact, today's process mining techniques are indeed able to support the whole BPM life cycle shown in Figure 2. Process mining is not only relevant for the design

and diagnosis/requirements phases, but also for the enactment/monitoring and adjustment phases.

Processes can be mined from several aspects, such as control-flow perspective, organizational perspective, case perspective or time perspective. This deliverable will focus on mining the control flow and the resources of a process. The next four sections will deal with how event logs are processed before process models are extracted out of them. Furthermore, the three types of mining will be discussed. Afterward, an approach to mine the organizational perspective of business processes (cf. Deliverable 2.1 Cabanillas et al. [2015b]) is described.

3.1 From event logs to process models

Figure 4 illustrates the typical information present in an event log used for process mining. The table shows events related to the handling of requests for an order fulfillment process. We assume that an event log contains data related to a single process. Moreover, each event in the log needs to refer to a single process instance, often referred to as case. In event logs, process instance executions are recorded as sequences of events, namely traces. Each trace, in turn contains a record of events. In Figure 4, each request corresponds to a case, e.g., Case 1, Case 2, etc. We also assume that events can be related to some activity. In Figure 4, events refer to activities like "Check stock availability", "Retrieve product from warehouse", etc. These assumptions are quite natural in the context of process mining. All mainstream process modeling notations, such as BPMN2 (OMG [2011]) or Petri Nets (Petri [1966]), specify a process as a collection of activities such that the life cycle of a single instance is described. Hence, the “case id” and “activity” columns in Figure 4 represent the bare minimum for process mining. Moreover, events within a case need to be ordered. For example, event "Ch-4680555556-1" (the execution of activity "Check stock availability" for Case 1) occurs before event "Re-5972222222-1" (the execution of activity "Retrieve product from warehouse" for the same case). Without ordering information, it is of course impossible to discover causal dependencies in process models.

Figure 4 also shows additional information per event. For example, all events have a timestamp (i.e., date and time information such as "2012-07-30 11:14"). This information is useful when analyzing performance related properties, e.g., the waiting time between two activities. The events in Figure 4 also refer to resources, i.e., the persons executing the activities. Also costs are associated to events. In the context of process mining, these properties are referred to as attributes.

Following up from Figure 4 the assumptions about an event logs are listed

Case ID	Event ID	Timestamp	Activity	Resource
1	Ch-4680555556-1	2012-07-30 11:14	Check stock availability	SYS1
1	Re-5972222222-1	2012-07-30 14:20	Retrieve product from warehouse	Rick
1	Co-6319444444-1	2012-07-30 15:10	Confirm order	Chuck
1	Ge-6402777778-1	2012-07-30 15:22	Get shipping address	SYS2
1	Em-6555555556-1	2012-07-30 15:44	Emit invoice	SYS2
1	Re-4180555556-1	2012-08-04 10:02	Receive payment	SYS2
1	Sh-4659722222-1	2012-08-05 11:11	Ship product	Susi
1	Ar-3833333333-1	2012-08-06 09:12	Archive order	DMS
2	Ch-4055555556-2	2012-08-01 09:44	Check stock availability	SYS1
2	Ch-4208333333-2	2012-08-01 10:06	Check materials availability	SYS1
2	Re-4666666667-2	2012-08-01 11:12	Request raw materials	Ringo
2	Ob-3263888889-2	2012-08-03 07:50	Obtain raw materials	Olaf
2	Ma-6131944444-2	2012-08-04 14:43	Manufacture product	SYS1
2	Co-6187615741-2	2012-08-04 14:51	Confirm order	Conny
2	Em-6388888889-2	2012-08-04 15:20	Emit invoice	SYS2
2	Ge-6439814815-2	2012-08-04 15:27	Get shipping address	SYS2
2	Sh-7277777778-2	2012-08-04 17:28	Ship product	Sara
2	Re-3611111111-2	2012-08-07 08:40	Receive payment	SYS2
2	Ar-3680555556-2	2012-08-07 08:50	Archive order	DMS
3	Ch-4208333333-3	2012-08-02 10:06	Check stock availability	SYS1
3	Ch-4243055556-3	2012-08-02 10:11	Check materials availability	SYS1
3	Ma-6694444444-3	2012-08-02 16:04	Manufacture product	SYS1
3	Co-6751157407-3	2012-08-02 16:12	Confirm order	Chuck
3	Em-6895833333-3	2012-08-02 16:33	Emit invoice	SYS2
3	Sh-7013888889-3	2012-08-02 16:50	Get shipping address	SYS2
3	Ge-7069444444-3	2012-08-02 16:58	Ship product	Emil
3	Re-4305555556-3	2012-08-06 10:20	Receive payment	SYS2
3	Ar-4340277778-3	2012-08-06 10:25	Archive order	DMS
4	Ch-3409722222-4	2012-08-04 08:11	Check stock availability	SYS1
4	Re-5000115741-4	2012-08-04 12:00	Retrieve product from warehouse	SYS1
4	Co-5041898148-4	2012-08-04 12:06	Confirm order	Hans
4	Ge-5223148148-4	2012-08-04 12:32	Get shipping address	SYS2
4	Em-4034837963-4	2012-08-08 09:41	Emit invoice	SYS2
4	Re-4180555556-4	2012-08-08 10:02	Receive payment	SYS2
4	Sh-5715277778-4	2012-08-08 13:43	Ship product	Susi
4	Ar-5888888889-4	2012-08-08 14:08	Archive order	DMS

Figure 4 Example of an event log taken from Dumas et al. [2013]

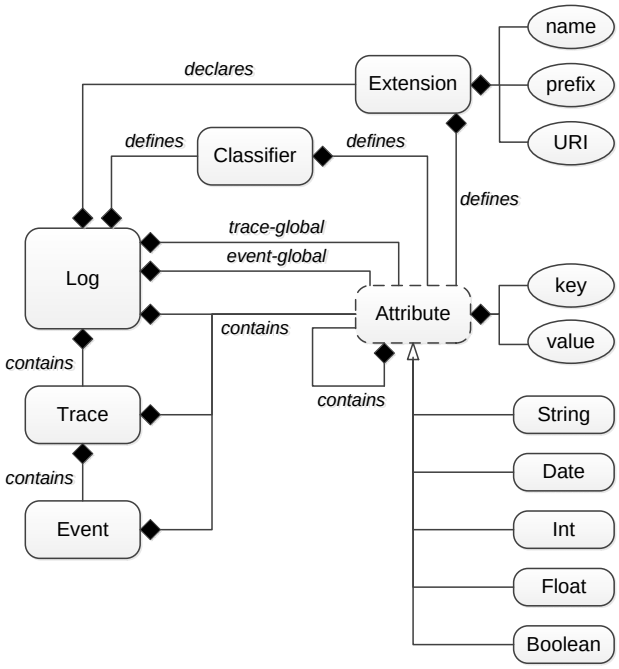


Figure 5 Example of the XES meta-model for logs as defined in Xes-standard.org [2015]. Picture taken from Dumas et al. [2013]

below.

- An event log consists of traces, each representing the execution of a process (case)
- A trace consists of events such that each event relates to precisely one case.
- Events within a case are ordered.
- Events can have attributes. Examples of typical attribute names are activity, time, costs, and resource

The de facto standard for storing and exchanging events logs is the XES format as defined in Xes-standard.org [2015].

XES may declare particular attributes to be mandatory. For example, it may be stated that any trace should have a name or that any event should have a timestamp. For this purpose, a log holds two lists of global attributes: one for the traces and one for the events. Users and organizations can add new extensions and share these with others. For example, general extensions referring to costs, risks, context, etc. can be added. However, extensions may also be domain specific (e.g., healthcare, customs, or retail) or organization specific. Currently, XES is supported by tools such as ProM (van Dongen et al. [2005]) and Disco[7].

[7] https://fluxicon.com/disco/

3.2 Process discovery

The first type of process mining is discovery. A discovery technique typically takes an event log and produces a model without using any a-priori information. An example is the α-algorithm (Van der Aalst et al. [2004]). This algorithm takes an event log and produces a Petri net explaining the behavior recorded in the log.

For example, given sufficient example executions of the process shown in Figure 4, the α-algorithm is able to automatically construct the Petri net without using any additional knowledge. If the event log contains information about resources, it is also possible to discover resource-related models, e.g., a social network showing how people work together in an organization.

3.2.1 The α-algorithm

Let us have a closer look at the α-algorithm. A detailed version of the algorithm can be found in Van der Aalst et al. [2004]. The α-algorithm is based on the following causality relations.

- $a >_L b$ if and only if there is a trace $\sigma = t_1, t_2, t_3, .., t_n$ and $i \in \{1, ..., n-1\}$
- $a \Rightarrow_L b$ if and only if $a >_L b$ and $b >_L a$
- $a \#_L b$ if and only if $a >_L b$ and $b >_L a$
- $a \|_L b$ if and only if $a >_L b$ and $b >_L a$

The above log-based ordering relations can be used to discover patterns in the corresponding process model as is illustrated in Figure 6.

The steps of the algorithm can be summarized as follows.

1. Identify the set of activities in the log as T_L
2. Identify starting activity as the first activity in each case T_I
3. Identify ending activity as the last activity in each case T_O
4. Identify the set of all connections to be potentially represented in the process model as a set X_L. Add the following elements to X_L.
   a. Pattern (a): all pairs for which hold $a \Rightarrow b$.
   b. Pattern (b): all triples for which hold $a \Rightarrow (b \# c)$.
   c. Pattern (c): all triples for which hold $(b \# c) \Rightarrow d$. Note that triples for which Pattern (d) or Pattern (e) hold are not included in X_L.
5. Construct the set Y_L as a subset of X_L by:
   a. Eliminating $a \Rightarrow b$ and $a \Rightarrow c$ if there exists some $a \Rightarrow (b \# c)$.
   b. Eliminating $b \Rightarrow c$ and $b \Rightarrow d$ if there exists some $(b \# c) \Rightarrow d$.
6. Connect start and end events in the following way:

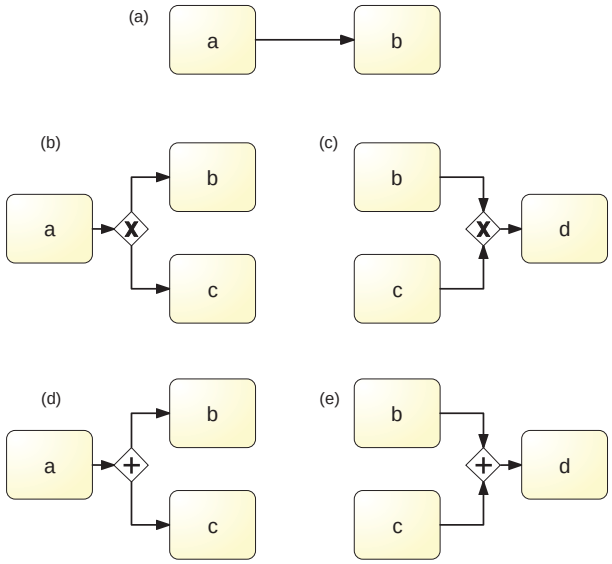


Figure 6 Patterns that can be discovered from the causality relations that are defined in the α-algorithm. Picture taken from Dumas et al. [2013]

a. If there are multiple tasks in the set T_I of first tasks, then draw a start event leading to an XOR-split, which connects to every task in T_I. Otherwise, directly connect the start event with the only first task.

b. For each task in the set T_O of last tasks, add an end event and draw an arc from the task to the end event.

Figure 7 shows the outcome of the α-algorithm on the log of Figure 4.

3.2.2 Advanced mining algorithms

The α-algorithm presented above is a nice introduction to process mining. It has scientific relevance, as it can be formalized in a compact way and some properties can be proofed. However, the α-algorithm can not deal with noise and incompleteness, and is not much used in practice. The following sections briefly overviews more advanced process discovery techniques.

3.2.2.1 Heuristic mining

Heuristic mining algorithms (**?**), take frequencies of events and sequences into account when constructing a process model. The basic idea is that infrequent paths should not be incorporated into the model. This applies normally to logs from real-world data, which are affected by noise and incompleteness. The output of a heuristic miner is a so-called Causal-net, which is a graph where nodes represent

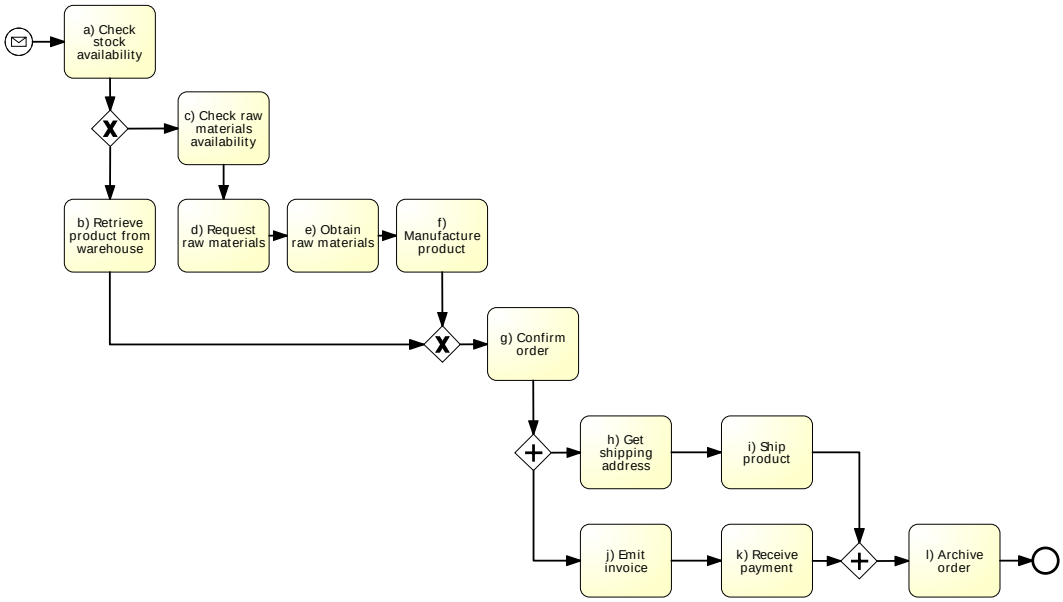


Figure 7 Result of the α-algorithm applied to the log of Figure 4. Picture taken from Dumas et al. [2013]

activities and arcs represent causal dependencies. Each activity has a set of possible input bindings and a set of possible output bindings. Causal nets produced by the heuristic miner are generally labeled with frequency values for each binding, that allow for identifying the mainstream behavior of the process.

3.2.2.2 Genetic process mining

Genetic process mining (**?**) adopts an evolutionary approach to mine processes out of data logs. The idea is to mimic the process of natural evolution. This approach is therefore not deterministic and depends on randomization to find new alternatives that fit best to some defined quality criteria. The algorithms consists of four main steps:

1. Initialization - The initial population, with process models as individuals, is randomly created and may only have a little to do with the data in the log.
2. Selection - The fitness value is computed for every individual. A fitness function takes into account the relation of the individual to the log, as well as criteria of log-fitness (i.e. ability to replay the log), simplicity, generalization and precision. Only the most fit individuals can move on from this stage to the following.
3. Reproduction - In this phase, parent individuals are selected to generate new processes. Two genetic operators are used: *crossover* and *mutation*. Crossover takes two individuals and creates new child process models that share genetic

material with their parents. These children are then modified through mutation. That is, random causal dependencies are added or removed from them. Mutation guarantees the evolution processes to go on beyond the initial genetic material.

4. Termination - The evolution process terminates when a satisfactory solution is found, i.e., a model having at least the desired fitness.

3.2.2.3 Region-based mining

Region-based mining algorithms allow to build up a Petri net from a transition system. First, events logs are transformed into transition systems where every position in each trace (i.e before the first event, in-between two events, or after the last event) is mapped into a state of a transition system. States can then be grouped together into Petri net places. Possible approaches for inducting places are based on:

- State-based regions - States from the transitions systems are grouped together based on regions with entering, leaving and non-crossing activities (**?**).
- Language-based regions - Regions are defined through behavioral observations on the log that correspond to a specified behavior. **?** use a linear programming reduction approach to identify minimal regions.

3.2.2.4 Fuzzy mining

? introduced *fuzzy mining* in order to deal with strongly unstructured real-world processes. While other process mining algorithms come out with models that show every relation they discover from the log, fuzzy miners able to abstract this data by preserving highly significant behavior, aggregating into clusters highly correlated behavior, and abstracting insignificant and lowly correlated behavior. This turns out very useful when it comes to analyzing complex spaghetti-like processes.

3.3 Process conformance

The second type of process mining is conformance. Here, an existing process model is compared with an event log of the same process. Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice-versa. For instance, there may be a process model indicating that purchase orders of more than one million Euro require two checks. The analysis of the event log can show whether this rule is followed or not. Another example is

the checking of the so-called "four-eyes" principle stating that particular activities should not be executed by one and the same person. By scanning the event log using a model specifying these requirements, one can discover potential cases of fraud. Hence, conformance checking may be used to detect, locate and explain deviations, and to measure the severity of these deviations. An example is the conformance checking algorithm described in Rozinat and van der Aalst [2008]. Given a process model and a corresponding event log, this algorithm can quantify and diagnose deviations.

3.4 Process enhancement

The third type of process mining is enhancement. Here, the idea is to extend or improve an existing process model using information about the actual process recorded in some event log. Whereas conformance checking measures the alignment between model and reality, this third type of process mining aims at changing or extending the a-priori model. One type of enhancement is repair, i.e., modifying the model to better reflect reality. For example, if two activities are modeled sequentially but in reality can happen in any order, then the model may be corrected to reflect this. Another type of enhancement is extension, i.e., adding a new perspective to the process model by cross-correlating it with the log (van der Aalst [2011a]). An example is the extension of a process model with performance data. For instance, by using timestamps in the event log of the "order fulfillment" process (see Figure 4), one can extend a log to show bottlenecks, service levels, throughput times, and frequencies. Similarly, logs can be extended with information about resources, decision rules, quality metrics, etc.

When extending process models, additional perspectives are added. Moreover, discovery and conformance techniques are not limited to control-flow. For example, one can discover a social network and check the validity of some organizational model using an event log. Hence, orthogonal to the three types of mining (discovery, conformance, and enhancement), different perspectives can be identified.

3.5 Resources and constraints

Two different types of processes can be distinguished Jablonski [1994]: well-structured routine processes with exactly predescribed control flow and agile processes with control flow that evolves at run time without being fully predefined a priori. Agile processes are common in healthcare where, e.g., patient diagnosis and treatment processes require flexibility to cope with unanticipated circumstances.

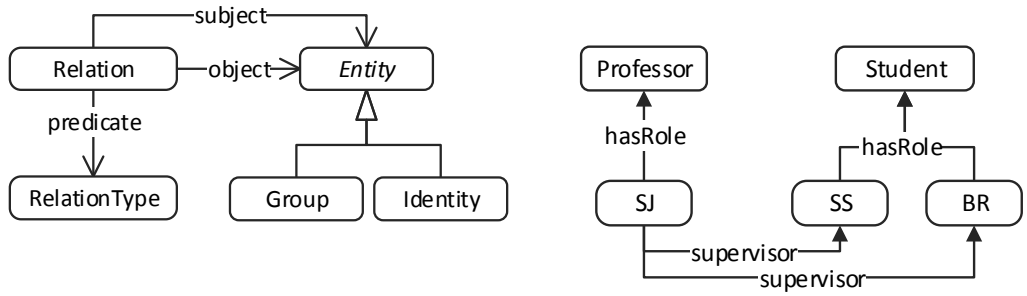
In a similar way, two different representational paradigms can be distinguished: procedural models describe which activities can be executed next and declarative models define execution constraints that the process has to satisfy. The more constraints are added to the model, the less possible execution alternatives remain. As agile processes may not be completely known a priori, they can often be captured more easily using a declarative rather than a procedural modeling approach van der Aalst et al. [2009], Pichler et al. [2012], Vaculín et al. [2011]. Agile processes need to explicitly integrate the organizational perspective in addition to the control flow perspective due to the importance of human decision-making and expert knowledge Marin et al.. Recent research has identified the potential of role mining Leitner et al. [2013], Baumgrass and Strembeck [2013] and process mining of the organizational perspective Zhao and Zhao [2014]. However, these results have not yet been integrated with declarative process models.

In the following we summarize a process mining approach to discover resource-aware, declarative process models that has been fully described in Schönig et al. [2015].

3.5.1 Selection of a Suitable Modeling Language

We choose *Declarative Process Intermediate Language (DPIL)* Zeising et al. [2014] for this purpose due to several reasons. First, it is multi-perspective, i.e., it allows representing several business process perspectives, namely, control flow, data and resources. Since we want to extract resource-aware process models, the modeling language needs to support the modeling of rules related to the organizational perspective. The expressiveness of DPIL and its suitability for business process modeling have been evaluated Zeising et al. [2014] with respect to the well-known Workflow Patterns Russell et al. [2005]. Second, it is multi-modal, meaning that it allows defining two different types of rules: rules representing mandatory relations (called *ensure* in DPIL) and rules representing recommended relations (called *advice* in DPIL). The latter are useful, e.g., to reflect good practices.

In order to express organisational relations, DPIL builds upon a generic organisational meta model that has been described in Bussler [1998] and is depicted in Figure 8a. It comprises the following elements: *Identity* represents agents that can be directly assigned to activities, i.e., both human and non-human resources. *Group* represents abstract agents that may describe several identities as a whole, e.g., roles or groups. *Relation* represents the different relations (*RelationType*) that may exist between these elements. It is suitable to define, e.g., that an identity has a specific role, that a person is the boss of another person, or that a person belongs



(a) Organisational meta model **(b)** Organisational model

Figure 8 Organizational meta model and example organizational model, taken from Schönig et al. [2015]

```
use group Professor

process BusinessTrip {
   task Book flight
   task Approve Application

   advice role(Approve Application, Professor)
   ensure sequence(Approve Application, Book flight)
}
```

Figure 9 Process for trip management modeled with DPIL

to a certain department. Figure 8b illustrates an exemplary organisational model of a university research group. It is composed of two roles (Professor, Student) assigned to three people (SJ, SS, BR) and several relations between them indicating who is supervised by whom.

DPIL provides a textual notation based on the use of *macros* to define reusable rules. For instance, the *sequence* macro ($sequence(a, b)$) states that the existence of a *start event* of task b implies the previous occurrence of a *complete event* of task a; and the *role* macro ($role(a, r)$) states that an activity a is assigned to a role r. Figure 9 shows an example of a process for trip management modelled with DPIL that uses the organizational model defined in Figure 8b. It states that it is mandatory to approve a business trip before a flight can be booked. Moreover, it is recommended but not necessary that the approval be carried out by a resource with the role *Professor*.

By applying process mining techniques, it is possible to generate process models like the one depicted in Figure 9, as long as the event logs contain the required information. Each event in a log refers to an activity, i.e., a well-defined step in the process, and is related to a particular process instance. Event logs usually come

with attribute that store information about the resource performing an activity van der Aalst [2011a], as well as additional information that may be useful for subsequent analysis purposes, such as start and/or completion time of activities for detailed temporal analyses. For instance, the following excerpt of a business trip process event log encoded in the XES logging format Verbeek et al. [2011a] shows the recorded information of the *start event* of an activity *Apply for trip* performed by a resource *SS*.

```
<event>
   <string key="org:resource" value="SS"/>
   <date key="time:timestamp" value="2013-08-06T14:58:00.000+01:00"/>
   <string key="concept:name" value="Apply for trip"/>
   <string key="lifecycle:transition" value="start"/>
</event>
```

3.5.2 Approach for Mining Resource-Aware Declarative Process Models

The approach will be described in four steps, namely, the generation and checking of rule candidates, a framework for the classification of rules, the definition of rule templates for mining the organisational perspective, and mechanisms for pre- and post-processing the data. It has been implemented and evaluated as described in Schönig et al. [2015].

3.5.2.1 Generation and Checking of Rule Candidates

Declarative process modelling languages like DPIL are based on so-called *rule templates*. A rule template captures frequently needed relations and defines a particular type of rules. Templates have formal semantics specified through logical formulae and are equipped either with user-friendly graphical representations (e.g., in Declare) or with macros in textual languages (e.g., in DPIL) that make the model easier to understand. In contrast to concrete rules, a rule template consists of placeholders, i.e., typed variables. It is instantiated by providing concrete values for these placeholders. For instance, the model described in Section 3.5.1 makes use of two rule templates represented by the macros sequence(T_1,T_2) and role(T,G). These templates comprise placeholders of type *Task* T as well as *Group* G. In all well-known declarative process mining approaches, rule templates are used for querying the provided event log and to find solutions to the placeholders. A solution to the query is any combination of concrete values for the placeholders that yields a concrete rule that is satisfied in the event log. First, all possible

rules need to be constructed by instantiating the given set of rule templates with all possible combinations of occurring process elements provided in the event log. The *sequence* template, e.g., consists of 2 placeholders of type *Task*. Assuming that $|T|$ different tasks occur in the event log, $|T|^2$ *rule candidates* are generated. All the resulting rule candidates are subsequently checked w.r.t. the event log.

In many cases, a rule candidate can be trivially valid. Consider, e.g., the rule candidate direct(t_1,i_1), i.e., start(of t_1) implies start(of t_1 by i_1), which holds when task t_1 is performed by an identity i_1, and the example event log of Table 2. The provided event log notation (first column) encodes the recorded start and complete events of a specific task t performed by an identity i with s(t,i) and c(t,i), respectively. The given events are ordered temporally so that timestamps are not encoded explicitly. In the first trace the rule holds trivially because t_1 never happens. Using the terminology of Maggi et al. [2012], we say that the rule is vacuously satisfied. It is necessary to discriminate between traces where a rule is trivially true and traces in which the rule is non-vacuously satisfied. Only traces in which a rule candidate non-trivially holds are considered interesting Maggi et al. [2011]. For first order logic rules that depict implications of the form $A \rightarrow B$ like in DPIL, trivially and non-vacuously valid rules can be discriminated by additionally checking the condition A of the rule separately.

Table 2 also shows the results of checking the non-vacuous satisfaction of the direct(t_1,i_1) rule (third column) as well as its condition (second column) for each trace of the example event log. In the first trace the rule is not (non-vacuously) satisfied because t_1 is never started, i.e., the condition is $false$. The rule holds non-vacously in the traces two to four while it is violated in trace five.

3.5.2.2 Support and Confidence Framework to Classify Rules

Checking rule candidates as described above provides for every candidate the number of instances, i.e., traces in the event log where it non-vacously holds. Based on these values it is possible to classify rules and to separate non-valid from valid ones. Therefore, Maggi et al. [2012] adopted a support and confidence

Trace	**start(of t_1)**	**direct(t_1,i_1)**
{**s**(t_2,i_1), **c**(t_2,i_2), **s**(t_3,i_1), **c**(t_3,i_1)}	false	false
{**s**(t_1,i_1), **c**(t_1,i_1), **s**(t_2,i_2), **c**(t_2,i_2), **s**(t_3,i_1), **c**(t_3,i_1)}	true	true
{**s**(t_1,i_1), **c**(t_1,i_1), **s**(t_3,i_3), **c**(t_3,i_3), **s**(t_2,i_2), **c**(t_2,i_2)}	true	true
{**s**(t_1,i_1), **c**(t_1,i_1), **s**(t_3,i_3), **c**(t_3,i_3), **s**(t_2,i_2), **c**(t_2,i_2)}	true	true
{**s**(t_1,i_4), **c**(t_1,i_4), **s**(t_3,i_1), **c**(t_3,i_1)}	true	false

Table 2 Event log and satisfaction of exemplary rule and its condition

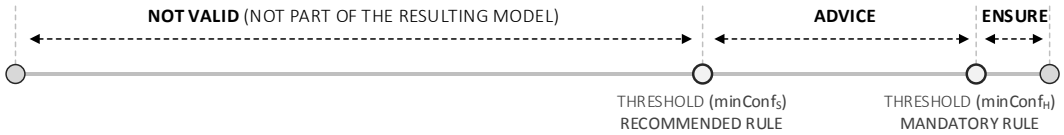


Figure 10 Classification of rule candidates based on the confidence value, taken from Schönig et al. [2015]

framework proposed by association rule mining methods to evaluate the relevance of rule candidates.

▶ **Definition 1** (Support and Confidence). Let $|\Phi|$ be the number of traces in an event log Φ. Let $|\sigma_{nv}(r)|$ be the number of traces in which a rule $r : A \rightarrow B$ is non-vacously satisfied. The support $supp(r)$ and confidence $conf(r)$[8] values of a rule r are defined as:

$$supp(r) := \frac{|\sigma_{nv}(r)|}{|\Phi|}, \qquad conf(r) := \frac{supp(r)}{supp(A)} \tag{1}$$

Considering the event log of Table 2 and the direct(t_1,i_1) rule, its support evaluates to $supp(r) = 0.6$ and its confidence to $conf(r) = 0.75$. The support value is used for pre-processing, as described in Section 3.5.2.4. We make use of the confidence value in order to classify a rule candidate r as (i) a *mandatory* rule, i.e., satisfied in almost all traces; (ii) a *recommended* rule, i.e., not always satisfied but with a tendency to be satisfied; or (iii) a *non-valid* rule, i.e., violated in most of the recorded traces. As visualized in Figure 10, two thresholds $minConf_S$ and $minConf_H$ are introduced to classify rule candidates. Candidates r with $conf(r) > minConf_H$ are classified as mandatory (*ensure*) and $minConf_S < conf(r) < minConf_H$ as recommended (*advice*). All rule candidates r with $conf(r) < minConf_S$ are non-valid rules and are not part of the resulting process model. Using the confidence values of rule candidates it is directly possible to generate a DPIL process model reflecting the recorded behavior.

3.5.2.3 Rule Templates for Analysing the Organisational Perspective

The previous section showed how it is possible to automatically generate a multi-modal, declarative process model by checking a set of rule candidates whose structure is defined by rule templates w.r.t. a given event log. Since DPIL builds upon a flexible organizational meta model, it is possible to define rule templates that define the structure of organizational relations. By instantiating these resource-aware

[8] In case of rules that do not depict implications, the condition is satisfied in every trace. Here $supp(A) = 1$ and $conf(r) = supp(r)$.

rule templates with all possible parameter combinations of defined resources, groups and relation types, it is possible to generate rule candidates that focus on the organizational perspective of the process to be analyzed. These candidates can then be checked under consideration of the corresponding organizational model. Based on the resulting confidence values, a resource-aware process model can be generated automatically.

We now define rule templates and their macros that can be used to mine the organizational perspective and to generate a resource-aware, declarative process model. We identified three different groups of organizational rule templates: resource allocation templates related to a single task, resource allocation templates related to more than one task, and cross-perspective rule templates, i.e., templates that express the influence of resources on the execution order of tasks. For every group we provide at least two representative examples that cover frequently needed organizational relations, according to the Workflow Resource Patterns Russell et al. [2005]. Nonetheless, further rule templates can be defined individually according to the analyst's needs.

We first focus on rule templates that define *resource allocation* patterns, i.e., rules that specify the resources which are allowed to perform a certain task. The *direct allocation* of resources to a task can be extracted by analyzing the *direct(T,I)* template. Given the free variables T and I and an event log with $|T|$ distinct tasks and $|I|$ distinct resources, there are $|T| \cdot |I|$ candidates to be checked.

```
direct(T,I) iff start(of T) implies start(of T by I)
```

Role-based allocation of resources can be identified with the *role(T,G)* template. Here, rule candidates for every task and group combination are generated, i.e., $|T| \cdot |G|$ rule candidates need to be checked.

```
role(T,G) iff start(of T by :p) implies
              relation(subject p predicate hasRole object G)
```

Organizational patterns can also relate to more than one task at the same time. The *binding(T_1,T_2)* template, e.g., can be used to discover if a task is always (mandatory) or should (recommended) be performed by the same resource as another task. With the *separate(T_1,T_2)* template, on the contrary, it is possible to discover task combinations that need to be or should be performed by different resources. For both templates, $|T|^2$ candidates need to be checked.

```
binding(T1,T2) iff start(of T1 by :p) and start(of T2) implies
                start(of T2 by p)
```

```
separate(T1,T2) iff start(of T1 by :p) and start(of T2) implies
                start(of T2 by not p)
```

The *orgDist(T_1,T_2,RT)* template is used to discover relations that are defined in the organizational model between the resources that performed two different tasks. This template incorporates a variable RT for the different relation types in the considered organizational model. Like this, $|T|^2 \cdot |RT|$ rule candidates exist.

```
orgDist(T1,T2,RT) iff start(of T1 by :p1) and start(of T2 by :p2) implies
                      relation(subject p1 predicate RT object p2)
```

Moreover, the organizational perspective can affect the execution order of tasks, i.e., the control flow of the process. There may be processes in which a sequence between tasks only holds for specific resources or for resources with a specific role. These patterns can be discovered by the *roleSequence(T_1,T_2,G)* and the *resourceSequence(T_1,T_2,I)* templates, respectively. For these templates, $|T|^2 \cdot |G|$ and $|T|^2 \cdot |I|$ candidates need to be checked.

```
roleSequence(T1,T2,G) iff start(of T2 by :p at :t) and
                      relation(subject p predicate hasRole object G)
                      implies complete(of T1 at < t)

resourceSequence(T1,T2,I) iff start(of T2 by I at :t) implies
                              complete(of T1 at < t)
```

3.5.2.4 Pre- and Post-processing

Real-life event logs and organizational models potentially contain a big set of distinct tasks, resources and groups. For instance, the BPI challenge 2011 event log of a hospital information system used by Bose and van der Aalst [2011] contains 623 different tasks and 42 organizational groups. By only considering the *role* template, this already leads to $623 \cdot 42 = 26166$ candidates to be checked. Although many of these parameter combinations never occur together in the same trace, the corresponding rules need to be checked. This problem also becomes obvious when considering task/resource combinations of the event log in Table 2. The resource i_4 only occurs together with task t_1. Hence, candidates of the *direct* template where $I = i_4$ and $T \neq t_1$ are trivially true in all traces and can be neglected without checking. The method proposed by Maggi et al. [2012] uses the well-known Apriori algorithm to pre-process the log and to extract *task combinations* that frequently occur together. A task combination is considered to be relevant if it occurs in a sufficient number of traces, i.e., above a given threshold $minSupp$. A $minSupp$ of 5%, e.g., claims that only rule candidates are considered

whose parameter combinations occur in at least 5% of the recorded traces. We extended this method in Schönig et al. [2014] to also extract *task/resource* and *task/-group* combinations that frequently occur together. This way, it is also possible to dramatically reduce the number of organizational rule candidates by abstracting from infrequent parameter combinations. Hence, for the example log, only 1 out of 3 *direct(T,i_4)* candidates are generated and checked.

Furthermore, when automatically generating a declarative process model, there are potentially extracted rules that are redundant. Consider, e.g., that a specific task t_1 has always been performed by a resource i_1 who has a role g_1 according to the organizational model. Then, the proposed method will (inevitably) discover a *role(t_1,g_1)* rule. This rule is redundant, since a direct allocation rule *direct(t_1,i_1)* will also be discovered. In case of i_1 *hasRole* g_1 the *role* rule is already implied in the *direct* rule. Redundant rules complicate the understandability of discovered models. Maggi et al. [2013] proposed a technique to post-process a discovered model and to remove redundant, *weaker* rules if they are already implied in *stronger* rules only focusing on the hierarchy of control flow perspective templates. We extended this method to also consider the rule hierarchies of organizational rules. Redundancy may also be caused by the interplay of three or more organizational rules. Consider, e.g., a set of discovered *binding* rules, such as *binding(t_1,t_2)*, *binding(t_2,t_3)* and *binding(t_1,t_3)*. Here, the rule between t_1 and t_3 is redundant because it belongs to the transitive closure of the other rules. In other words, if task t_1 has always been performed by the same resource as t_2, and t_3 has always been performed by the same resource as t_2, then also t_1 and t_3 have been performed by the same resource. Not all rule types can be reduced using transitive reduction. *Separate* rules, e.g., are not transitive, i.e., if t_1 is not performed by the same resource as t_2, and t_2 is not executed by the same resource as t_3, then we cannot conclude automatically that t_1 is also not performed by the same resource as t_3.

4 Mining Project-Oriented Business Processes

SHAPE aims to provide ICT support for more rigorous and verifiable process management in such recurring and adaptive engineering processes: SHAPE shall support monitoring and conformance checking in safety-critical engineering processes. Process monitoring can be supported through existing documentation that can be found in project repositories. Data includes manually compiled handbooks, emails, diagrams and several files that are used by specific purpose tools. These data are typically stored in VCS. Such systems are an essential means for large collaborative projects that requires for data safety, version tracking and coordination

among teams.

A requirement in the railway domain is to monitor if processes are executed according to existing norms and regulations. Processes need to be documented in all their steps, in order to allow for backtracking. However, when no process engine to support execution is used, process history can be obtained from VCS logs. Such logs can reflect the work history that was done in the work packages. Assuming that projects are organized according to a meaningful structure in the VCS, i.e. directories are subdivided into folders data and contribution from each work package is stored in a corresponding folder, it is possible to mine the logs to get relevant information about the progress of the work stream over time.

A relevant class of business processes in the railway domain are ad-hoc planned ones from expert engineers to respond to domain necessities. They are executed only once with limited resources and budget to attain a predetermined goal. We refer to this class as *project-oriented business processes*. In this work we want to extract these type of processes from VCS log data and display them as Gantt charts.

The following sections give a background on what already exists in literature to tackle this problem. Then we present a method to extract such processes.

4.1 Literature review

Two main directions have been adopted so far in literature to address the aforementioned problem. The first class strives to transform VCS log data into logs that comply to the XES meta model for process mining as defined in Verbeek et al. [2011a]. Several approaches have been developed to preprocess VCS data such that process mining techniques can be applied, and hence, a business process can be derived from the log data. In this group, Kindler et al. [2006a,b] developed an algorithm for extracting software processes that are mapped to Petri Nets. Activities, which are not explicit in the logs, are discovered from their input and output artifacts. However, strong assumptions are made on the filenames as well as on the software process lifecycle. Rubin et al. [2007] addressed the problem of engineering processes that are not well documented and are usually unstructured. They provided a bridge from Kindler et al. [2006a,b]'s approach to ProM (van Dongen et al. [2005]) in order to mine different process perspectives, such as performance social network analyzes. Rubin et al. [2014] applied process mining to the touristic industry and obtained user processes from web client logs pursuing the goal of improving the software system by analyzing the underlying process. Poncin et al. [2011] developed the FRASR framework for preprocessing software repositories to transform the VCS data to logs that conform to the process mining event log

meta model (van Dongen and Van der Aalst [2005]) as utilized in ProM. However, these approaches disregard the single-instance nature of project-oriented business processes and treat them as procedures that can be repeated over time.

The second category of related work focuses on the visualization of VCS data for different purposes.

Several approaches study the interaction among developers over time from a visualization point of view. For instance, Ogawa and Ma [2010] drew storyline pathways to show the story of each developer's contribution. Other approaches analyze and visualize VCS data at file level in order to discover file version evolution. Voinea and Telea [2006] introduced an interactive navigation method to surf file version evolution as well as two methods to cluster versions of the same file in an abstraction layer. Wu et al. [2004] also visualized the evolutions of entire projects at file level, emphasizing the evolution moments. Finally, several approaches study change prediction with the aim of discovering prediction patterns that can help in the process of software development (Zimmermann et al. [2004], Ying et al. [2004]). The approaches mentioned in this category as well as others that apply similar techniques (Feldt et al. [2013], Kagdi et al. [2006], D'Ambros and Lanza [2008]) focus on studying software evolution from different standpoints. However, the goal pursued differs in all cases from our goal in that they are not interested in discovering projects tasks out of the log data, and hence, they lack an explicit notion of work structure that we need to consider for our purpose.

4.2 Project-oriented processes discovery method

This section presents a discovery methods for project-oriented business processes. Formalisms will be omitted in order to have a better overview of the technique. [9]

Figure 11 shows the context of our project mining technique. At the top level there are people that work together in a project. These people commit their work progress using Version Control Systems (VCSs) like Subversion (Pilato et al. [2008]) or Git (Torvalds and Hamano [2010]). Each resource can work on multiple files. Thus, each commit contains an undefined number of events. Events are defined as atomic changes on a file or directory.

[9] For full details on the defined formalisms about mining project-oriented processes, please refer to Mining Project-Oriented Business Processes paper that was submitted to BMP15. The paper can be found on https://ai.wu.ac.at/svn/SHAPE_FFG_845638/WP3/Papers/project%20mining/project%20mining-submitted-v3.pdf

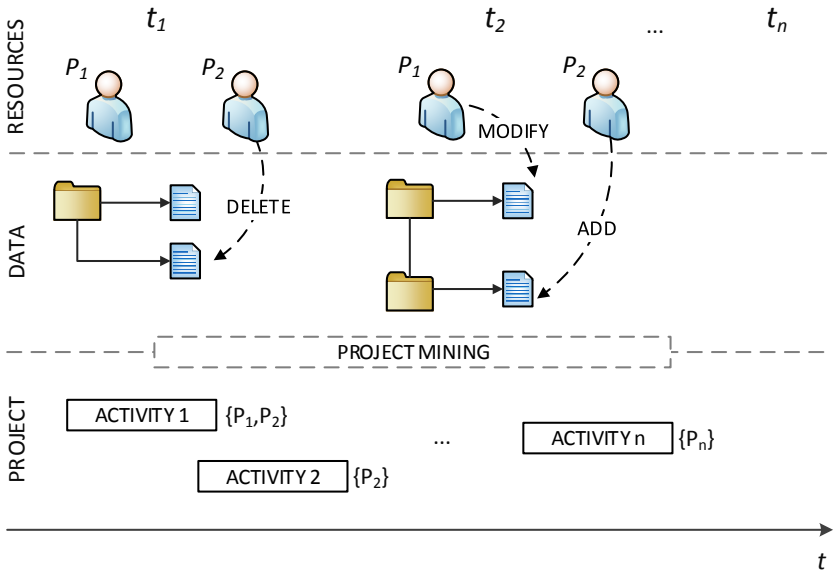


Figure 11 Project mining scenario

Projects are decomposed into work packages. We assume a hierarchical work package structure of a project, such that a work package can have sub work packages. Further, the amount of work in a single work package need not be done in one single time span, but it can be split into several activities. Activities have a start and end time, and subsequent activities can have idle periods in between. Thus, we define projects as follows.

▶ **Definition 2** (Project). A project P is a tuple $(W, S, A, \alpha, \omega, \beta)$, where

- W is the set of work packages in the project.
- $S \subseteq W \times W$ is the relation that hierarchically decomposes work packages into a tree structure.
- A is the set of activities that are conducted in the work packages.
- $\alpha : A \to TS$ is the function that assigns a start time to activities. Activities are ordered by their start times.
- $\omega : A \to TS$ is the function that assigns an end time to activities.
- $\beta : A \to W$ is the mapping function that maps activities to their corresponding work packages.

Project mining is defined as the technique that discovers a project P from event log data.

For project discovery from the VCS commit history, we need to identify activities that are performed, associate the activities to work packages and recreate the work package structure of the project. Our aim is to create a hierarchical model that provides an overview of the project work. Therefore, we have to identify the

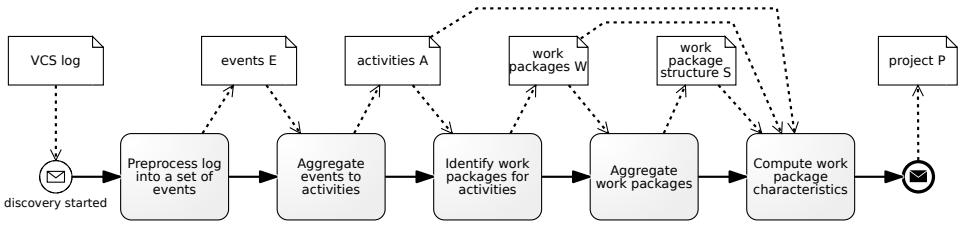


Figure 12 Project discovery technique overview as BPMN process model.

start and end times of activities and of work packages before we can visualize the project work. The input to the technique is the log that is stored in the VCS. The challenge is that the raw log only records commits on the file system level and information on activity level is missing. However, we can deduce activity information from events based on the following assumptions.

A1: Meaningful file tree structure. The file tree structure in a project represents its work package structure. That is, the knowledge workers organize their work in a file hierarchy that reflects the project structure.

A2: Local changes. Activities in a work package affect only files of the work package folder, or in the corresponding sub-tree in the file tree structure.

A3: Frequent commits. Commits to the VCS are regularly performed, when conducting work in an activity.

Note that assumption A1 can be seen as a strong assumption on the file tree structure. Nevertheless, we argue that even if A1 is not entirely met, the aggregation of work information on the file tree hierarchy provides a valuable view on the project.

Figure 12 illustrates the different steps of the technique. We describe each of them in detail.

4.2.1 Step 1: Preprocessing

The first step is to transform raw logs of the VCS (which might be grouped by commits) into a list of events. This is done by replicating the information on commit level to be contained in the events. At the same time, the directory structure is reconstructed from the logs. This is done by extracting all the changed file paths that are stored in the different entries of the log. The output of this step is a set E of events associated to each object of the directory structure.

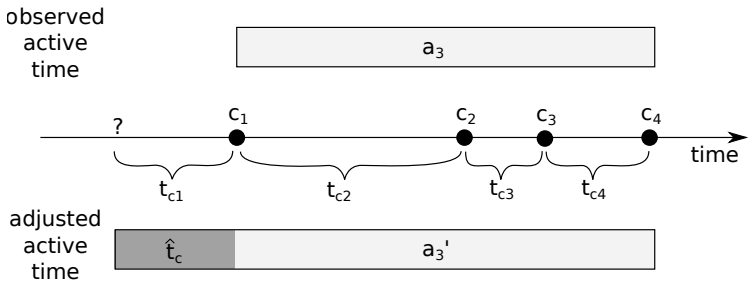


Figure 13 Adjustment of activity start time by the expected time before commit.

4.2.2 Step 2: Aggregating events to activities

Given the set of events that we gathered in Step 1 from a VCS, the next step is to identify the activities to which the events belong. Note that we do not know the activities of the project in advance, but need to infer them based on the events. Each event affects a single file in the file hierarchy.

Based on assumption A2, we are interested in activities conducted in a work package, that is, we filter for the events that are contained in the given file or its children files in the hierarchy. For every file of interest, we select the set of events affecting the file or its children. The next step is then to find the activities which emitted the set of events of the file. We rely on assumption A3, which states that during an activity, we expect multiple commits. Assumption A3 allows us to conclude that if we do not observe commits for a longer period of time, there is no activity being performed in the work package.

We adopt the abstraction technique by [Baier et al., 2014] and allow the domain expert to formulate rules for aggregating events to activities based on boundary conditions. Assuming that people frequently commit their progress (A3), we can specify a boundary condition based on the temporal distance to previous events. For example, we can specify that a time period of seven days without a commit is a boundary condition. As a result, we obtain the mapping from events to these activities. The set of discovered activities identified for the work package based on given boundary conditions is then made of groups of events that happened within seven days from each other.

With the events mapped to activities, we need to find the temporal boundaries of the target activities. The challenge here is that we do not know when an activity actually started, because the start of the activity is not recorded in the VCS. We can only observe the time of the first commit in that activity, but commits usually mark progress of an already running activity.

To address the challenge of missing start times, we impute the missing start time by prepending the expected active time $\hat{t}_c$ before a commit, as illustrated

by Figure 13. This notion assumes that project participants commit their work progress after a certain amount of time. However, we cannot compute $\hat{t}_c$ by looking at the average commit rate in a work package, because this average is based on busy periods and idle periods. We need to factor out the idle periods in the computation of this measure.

We know the end time of the activities, as the last commit marks the completion of work. Therefore, each activity a based on given boundary conditions has the associated end time equal to the highest time value of its events. We call it ω (a). Further, we write the lowest timestamp value of the events of a as α'(a). Let c (a) be the number of commits in a. Let A_f be the set of activities that correspond to file $f \in F$, the expected active time between commits $\hat{t}_c$ is given as follows.

$$\hat{t}_c = \frac{\sum_{a \in A_f} (\omega(a) - \alpha'(a))}{\sum_{a \in A_f} (c(a) - 1)} \tag{2}$$

The assumption is that is at least one activity spanning over at least two commits. Translated to our boundary condition, this assumption is that there is at least one week in each work package, in which there were at least two commits made. Otherwise, we set $\hat{t}_c$ to 0 for the current file due to lack of information.

Given the expected active time between commits $\hat{t}_c$, we can finally adjust the start time of each activity. Therefore, we set the associated start time for each activity as $\alpha(a) = \alpha'(a) - \hat{t}_c$. That is, we subtract the expected active time from the first commit's timestamp.

We apply Step 2 to all files in the file tree to get the activities per file. The activities A of the project are obtained as the union of the activity sets per file $\bigcup_{f \in F} A_f$.

4.2.3 Steps 3 and 4: Mapping activities to work packages and aggregating.

Once activities have been identified, we want to climb to the next abstraction layer: the work packages. Assumption A1 allows us to specify a one-to-one mapping $\kappa : F \to W$ between files in the file tree structure and work packages. More precisely, we construct the set of work packages W isomorphic to the set of files F, such that the *Parent* relation is preserved in the work package structure S relationship.

The mapping β of activities to work packages is simply $\beta(a) = \kappa(\psi(a))$, where $\psi(a)$ is defined as the function $\psi : A \to F$ that contains the mapping information of the discovered activities to their originating files. In this way, we provide an

activity based view on work packages, and we can aggregate on each level in the file system to see active periods of the corresponding hierarchy level.

4.2.4 Step 5: Computing work package characteristics.

In this final step, we compute measures of interest for the discovered work packages. First, we obtain the temporal boundaries of a work package by the functions α and ω of the associated activities.

The start and end time of a work package (α_W and ω_W) are functions from work packages to timestamps. The start time is defined as a the least timestamp of the set of activities' starting times $\alpha(a)$ that compose the workpackage w. Analogously, the end time function of work packages is defined as the maximum of the end times $\omega(a)$ of the activities of w. The duration of a work package τ is the difference between ω_W and α_W.

We want to estimate the average work intensity in a work package. Therefore we define the following concept of *coverage* in Definition 3.

▶ **Definition 3** (Coverage)**.** The coverage χ of work packages by activities is a function $\chi : W \to [0, 1]$ and is defined as follows.

$$\chi(w) = \frac{\sum_{a \in \beta^{-1}(w)} (\omega(a) - \alpha(a))}{\tau(w)} \tag{3}$$

With this final step, we lifted the information hidden in low level events to a high-level Gantt chart perspective, with which project managers are familiar.

4.3 Project visualization

This section shows how the tool visualizes the projects. We apply our algorithm to an example case from Table 3 and check how it helps to identify work packages. The data is aggregated according to our threshold of seven days. We can observe three groups of events being temporally close to each other according to our threshold. That is, we expect the event data to be grouped into three activities.

The second step of our algorithm takes care of adjusting the starting time of the activities. Furthermore, we vertically order the events and activities in the Gantt chart according to the directory structure to show the mapping from the objects on the Gantt chart to each work package in the tree structure. The last step, computing work package characteristics, is done automatically when we collapse a node on of tree.

Table 3 Excerpt from the Toy Example VCS log data taken from SHAPE

CID	Resource	Date	List of changes
1	Y	2014-11-12 11:57:46	A /example A /example/SHAPE/ToyStation-Example.docx
...	...	...	...
3	X	2014-11-14 16:34:07	M /example/ToyStation.bpmn M /example/ToyStation.png
4	W	2014-12-15 13:49:11	D /example/Download
5	W	2015-01-08 16:06:41	A /example/Download2
6	X	2015-01-13 11:47:09	M /example/ToyStation_0Loop.bpmn M /example/ToyStation_nLoop.bpmn
7	Z	2015-01-16 16:50:29	A /example/ToyStation_0Loop.pdf A /example/ToyStation-feedbackZ.pdf

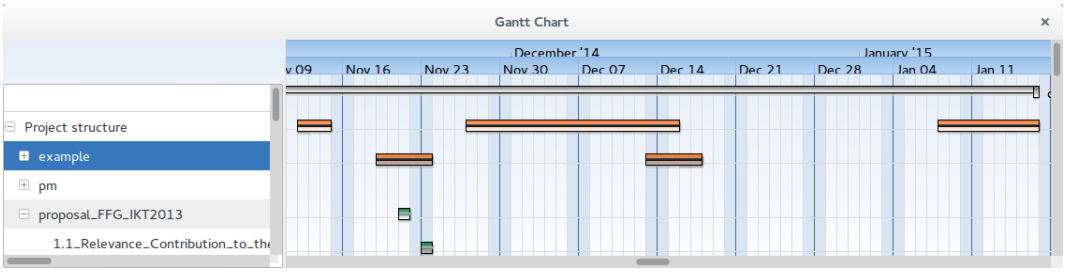


Figure 14 Visualizing events and activities

4.4 Coverage tests

Table 4 shows coverage tests for open source projects and projects from SHAPE. We are interested in exploring how the coverage factor varies in different existing projects. Hence, we take the work package w as our controlled variable and set it to the highest level of aggregation. Then, we analyze each project of the data set and observe the dependent variable $\chi(w)$. Another variable of interest is the $\hat{t}_c$ since it gives an idea of the average work speed (commit frequency) during active times.

The data we used stems from the following projects. *MiningVCS* is our tool. It consists of daily commits and was developed over 24 days. *Whitehall* is the code name for the Inside Government project, which aims to bring Government departments online in a consistent and user-friendly manner. *Petitions* is a Drupal 7 code base used to build an application on "We The People", the platform to create and sign petitions of the White House. *Study* is an SVN log about Healthcare domain, taken from SHAPE. *The guardian* is the log data from the Git repository of the well-known British national daily newspaper. *Book* is the log data that

Table 4 Coverage results for different open source projects

Log File name	Duration Days	Idle periods Number	Files Number	Commits Number	$\hat{t}_c$ Hours	$\chi(w)$ %
MiningCVS	24	0	89	63	9	100
Whitehall	1279	6	6539	15566	2	95
Petitions	834	17	1562	914	13	59
Study	624	13	7501	736	11	58
The Guardian	1667	59	12889	621	30	44
Book	414	15	154	592	5	32
Papers	1859	55	1791	649	20	30
Requirements	771	22	505	231	17	21
Yelp	206	6	24	54	20	20
Adobe	1076	13	356	237	24	15

describes the writing of the book Crypto 101 by Laurens Van Houtven, taken from Git. *Papers* is taken from SHAPE project for building a paper archive. *Requirements* log data is taken from the the Git repository of OpenETCS and belongs to the railway domain. *Yelp* is the main Github page of Yelp were they showcase all their projects. *Adobe* is the Adobe Github Homepage v2.0, which is a central hub for Adobe Open sources projects.

Table 4 shows our experiments on the above-mentioned logs and the corresponding coverage factors. Projects that score a high coverage factor are characterized by continuous work. This can be further seen by looking at their average idle times $\bar{t}_{Idle}$. Let n_c be the number of commits per work package. We compute the average idle time as follows.

$$\bar{t}_{Idle} = \frac{\tau - n_c \cdot \hat{t}_c}{n} \ , \ n > 0 \tag{4}$$

where n is the number of idle times in the work package. If $n = 0$, then we trivially assign $\bar{t}_{Idle} = 0$, because there were no break periods over time.

Applying the formula to the above projects, we can observe how projects with a higher coverage factor have actually low values of $\bar{t}_{Idle}$. For instance, *Whitehall* scores a $\bar{t}_{Idle}$ of 11 days, whereas *Adobe* scores a $\bar{t}_{Idle}$ of 36 days. This supports the usage of the coverage factor χ as an indicator for work package time utilization.

4.5 Comparison with approaches that use classical process mining

In this section we compare our method to other alternatives for mining data out of logs and interpret our results.

Well known tools that are used in academia and practice include ProM [van Dongen et al., 2005] and Disco[10]. Both tools require input data to be in the XES [Verbeek et al., 2011a] format. Thus, we convert our data from the *Define example* case into XES. To show events per objects of the project structure, we choose the file path as the *caseId*. To flatten the logs we extract all the file paths and build a mapping from each file to the set of changes done to it.

Figure 15 depicts the results of the Dotted chart plugin of ProM applied to our log data. Also here, we observe different changes of each file of the repository. While the files and their corresponding events are shown, the plugin does not allow to rearrange the data in order to understand the file structure, nor does it allow to perform any kind of aggregation or connection between data, to observe them from a higher level perspective.

Figure 16 shows the results from mining our log data with the Disco tool. Here we can see a plot that displays the events that happen over time. The plot has some peeks in correspondence to active times of the *example* work package. They can be grouped in three clusters: an initial cluster with a few amount work, an intermediate cluster with the most significant part of the work, and a final cluster that again is not very active. In this way, clusters can be associated to activities. As a drawback, when the number of work packages and activities increase, the number of peeks grows and generate identifying clusters of activities by look at active (or idle) times becomes unworkable.

Our approach to mining the work progress of project-oriented business processes complements these techniques with metrics and a corresponding visualization that is informative to managers.

5 Summary

As the goal of WP3 is to discover existing processes with the final objective of supporting conformance checking, in this deliverable we introduced the fields of text mining and process mining. We showed what problems they allow addressing. Text mining is useful when dealing with unstructured data. Using information ex-

[10] http://fluxicon.com/disco/

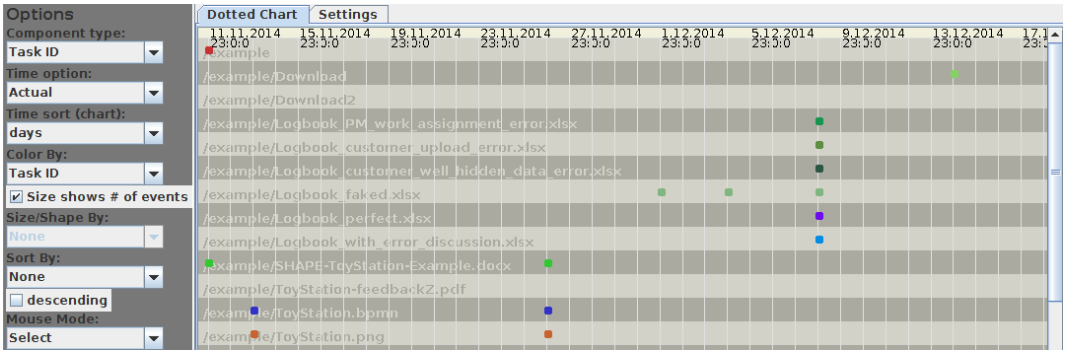


Figure 15 Dotted chart from ProM

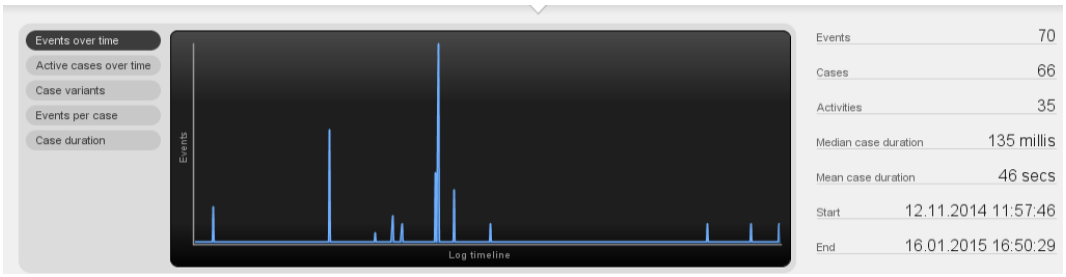


Figure 16 Chart from Disco plotting the events over time.

traction techniques it is possible to understand entities and relations from textual documents that are filled out by engineers. Using NLP techniques it is possible to get a more precise understanding of the semantics of the textual content of the documents.

On the other hand, process mining can be used with more structured documents, specifically with logs that store data according to the process mining meta-model as defined in van Dongen and Van der Aalst [2005]. In such cases, it is possible to transform the logs into the XES format and apply well-known process mining techniques to gather information on the control flow. Agile processes make an intense use of resources. In this case, mining the organizational perspective with declarative modeling languages provides valuable information.

Furthermore, we showed a work that mines a particular type of processes, referred to as *project-oriented* business processes. These processes come from logs that do not adhere to the process mining meta-model. Our method allows for visualizing and navigating the processes as GANTT charts and it also provides information about the work effort.

We intend to combine the methods described in this deliverable in order to analyze and use information from both structured and unstructured data sources. In the near future we aim at extending our algorithm for mining project-oriented business processes to take into account the comments that may be present in the

VCS logs in order to improve the discovery of activities.

References

C.C. Aggarwal and C.X. Zhai. *Mining Text Data*. Springer, 2012. ISBN 9781461432234. URL http://books.google.at/books?id=vFHOx8wfSU0C.

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

James Allan, Jaime G Carbonell, George Doddington, Jonathan Yamron, and Yiming Yang. Topic detection and tracking pilot study final report. 1998.

Thomas Baier, Jan Mendling, and Mathias Weske. Bridging abstraction layers in process mining. *Information Systems*, 46:123–139, 2014.

Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction for the web. In *IJCAI*, volume 7, pages 2670–2676, 2007.

Anne Baumgrass and Mark Strembeck. Bridging the gap between role mining and role engineering via migration guides. *Inf. Sec. Techn. Report*, 17(4):148–172, 2013. 10.1016/j.istr.2013.03.003. URL http://dx.doi.org/10.1016/j.istr.2013.03.003.

RP Jagadeesh Chandra Bose and Wil MP van der Aalst. Analysis of Patient Treatment Procedures. In *Business Process Management Workshops*, pages 165–166, 2011.

Christoph Bussler. *Organisationsverwaltung in Workflow-Management-Systemen*. Dt. Univ.-Verlag, 1998.

Cristina Cabanillas, Alois Haselböck, Jan Mendling, Axel Polleres, Simon Sperl, and Simon Steyskal. Engineering Domain Ontology: Base Regulations and Requirements Description. Project deliverable, Vienna University of Economics and Business, Austria, 2015a.

Cristina Cabanillas, Giray Havur, Jan Mendling, Axel Polleres, and Alexander Wurl. State-of-the art on existing models for processes, resources, constraints and security, and their underlying formalisms. Project deliverable, Vienna University of Economics and Business, Austria, 2015b.

Jim Cowie and Wendy Lehnert. Information extraction. *Communications of the ACM*, 39(1):80–91, 1996.

M. D'Ambros and M. Lanza. A Flexible Framework to Support Collaborative Software Evolution Analysis. In *Software Maintenance and Reengineering (CSMR'08)*, pages 3–12, April 2008. 10.1109/CSMR.2008.4493295.

Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A Reijers. *Fundamentals of business process management*. Springer, 2013.

Robert Feldt, Miroslaw Staron, Erika Hult, and Thomas Liljegren. Supporting software decision meetings: Heatmaps for visualising test and code measurements. In *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*, pages 62–69. IEEE, 2013.

Vishal Gupta and Gurpreet S Lehal. A survey of text mining techniques and applications. *Journal of emerging technologies in web intelligence*, 1(1):60–76, 2009.

Stefan Jablonski. MOBILE: A modular workflow model and architecture. In *Working Conference on Dynamic Modelling and Information Systems*, 1994.

Thorsten Joachims. *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.

Dan Jurafsky and James H Martin. *Speech and language processing*. Pearson, 2014.

Huzefa Kagdi, Shehnaaz Yusuf, and Jonathan I. Maletic. Mining Sequences of Changed-files from Version Histories. In *Workshop on Mining Software Repositories (MSR '06)*, pages 47–53. ACM, 2006. 10.1145/1137983.1137996.

Boris Katz. From sentence processing to information access on the world wide web. In *AAAI Spring Symposium on Natural Language Processing for the World Wide Web*, volume 1, page 997. Stanford University Stanford, CA, 1997.

Ekkart Kindler, Vladimir Rubin, and Wilhelm Schäfer. Activity Mining for Discovering Software Process Models. *Software Engineering*, 79:175–180, 2006a.

Ekkart Kindler, Vladimir Rubin, and Wilhelm Schäfer. Incremental Workflow Mining Based on Document Versioning Information. In Mingshu Li, Barry Boehm, and LeonJ. Osterweil, editors, *Unifying the Software Process Spectrum*, volume 3840 of *Lecture Notes in Computer Science*, pages 287–301. Springer Berlin Heidelberg, 2006b. ISBN 978-3-540-31112-6. 10.1007/11608035_25. URL http://dx.doi.org/10.1007/11608035_25.

Maria Leitner, Anne Baumgrass, Sigrid Schefer-Wenzl, Stefanie Rinderle-Ma, and Mark Strembeck. A case study on the suitability of process mining to produce current-state rbac models. In *Business Process Management Workshops*, pages 719–724, 2013.

Fabrizio Maria Maggi, Arjan Mooij, and Wil van der Aalst. User-Guided Discovery of Declarative Process Models. In *Computational Intelligence and Data Mining*, pages 192–199, 2011.

Fabrizio Maria Maggi, Jagadeesh Chandra Bose, and Wil van der Aalst. Efficient Discovery of Understandable Declarative Process Models from Event Logs. In *Advanced Information Systems Engineering*, pages 270–285, 2012.

Fabrizio Maria Maggi, Jagadeesh Chandra Bose, and Wil van der Aalst. A Knowledge-Based Integrated Approach for Discovering and Repairing Declare

Maps. In *Advanced Information Systems Engineering*, pages 433–448, 2013. URL http://link.springer.com/chapter/10.1007/978-3-642-38709-8_28.

Mike Marin, Richard Hull, and Roman Vaculín. Data Centric BPM and the Emerging Case Management Standard : A Short Survey Case Management. 257593.

R Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 328–334, 1999.

Michael Ogawa and Kwan-Liu Ma. Software evolution storylines. In *Proceedings of the 5th international symposium on Software visualization*, pages 35–42. ACM, 2010.

OMG. BPMN 2.0. Recommendation, OMG, 2011.

Carl Adam Petri. Communication with automata. 1966.

Paul Pichler, Barbara Weber, Stefan Zugal, Jakob Pinggera, Jan Mendling, and Hajo Reijers. Imperative versus declarative process modeling languages: An empirical investigation. *Business Process Management Workshops*, pages 383–394, 2012.

C Michael Pilato, Ben Collins-Sussman, and Brian W Fitzpatrick. *Version control with subversion*. "O'Reilly Media, Inc.", 2008.

Wouter Poncin, Alexander Serebrenik, and Mark van den Brand. Process mining software repositories. In *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*, pages 5–14. IEEE, 2011.

Anne Rozinat and Wil MP van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.

Vladimir Rubin, Christian W Günther, Wil MP Van Der Aalst, Ekkart Kindler, Boudewijn F Van Dongen, and Wilhelm Schäfer. Process mining framework for software processes. In *Software Process Dynamics and Agility*, pages 169–181. Springer, 2007.

Vladimir Rubin, Irina Lomazova, and Wil MP van der Aalst. Agile development with software process mining. In *Proceedings of the 2014 International Conference on Software and System Process*, pages 70–74. ACM, 2014.

Nick Russell, Wil MP van der Aalst, Arthur HM Ter Hofstede, and David Edmond. Workflow resource patterns: Identification, representation and tool support. In *Advanced Information Systems Engineering*, pages 216–232, 2005.

Stefan Schönig, Florian Gillitzer, Michael Zeising, and Stefan Jablonski. Supporting rule-based process mining by user-guided discovery of resource-aware frequent patterns. In *ICSOC 2014 Workshops, in press*, 2014.

Stefan Schönig, Cristina Cabanillas, Stefan Jablonski, and Jan Mendling. Mining the Organisational Perspective in Agile Business Processes. In *BPMDS*, page In press., 2015.

Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

Kristie Seymore, Andrew McCallum, and Roni Rosenfeld. Learning hidden markov model structure for information extraction. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, pages 37–42, 1999.

Linus Torvalds and Junio Hamano. Git: Fast version control system. *URL http://git-scm. com*, 2010.

Roman Vaculín, Richard Hull, Terry Heath, Craig Cochran, Anil Nigam, and Piyawadee Sukaviriya. Declarative business artifact centric modeling of decision and knowledge intensive business processes. In *EDOC*, pages 151–160, 2011.

Wil van der Aalst. *Process mining: discovery, conformance and enhancement of business processes*. 2011a.

Wil Van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *Knowledge and Data Engineering, IEEE Transactions on*, 16(9):1128–1142, 2004.

Wil van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23(2):99–113, 2009.

Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes.* Springer, 2011b. ISBN 978-3-642-19344-6.

Boudewijn F van Dongen and Wil MP Van der Aalst. A Meta Model for Process Mining Data. *EMOI-INTEROP*, 160:30, 2005.

Boudewijn F van Dongen, Ana Karla A de Medeiros, HMW Verbeek, AJMM Weijters, and Wil MP Van Der Aalst. The ProM framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets 2005*, pages 444–454. Springer, 2005.

Eric Verbeek, Joos Buijs, Boudewijn van Dongen, and Wil van der Aalst. XES, xESame, and ProM 6. In *Information Systems Evolution*, pages 60–75, 2011a.

HMW Verbeek, Joos CAM Buijs, Boudewijn F Van Dongen, and Wil MP Van Der Aalst. Xes, xesame, and prom 6. In *Information Systems Evolution*, pages 60–75. Springer, 2011b.

Lucian Voinea and Alexandru Telea. Multiscale and Multivariate Visualizations of Software Evolution. In *Symposium on Software Visualization (SoftVis'06)*, pages 115–124. ACM, 2006. 10.1145/1148493.1148510.

Charles L Wayne. Multilingual topic detection and tracking: Successful research enabled by corpora and evaluation. In *LREC*, 2000.

Ian H. Witten, Zane Bray, Malika Mahoui, and Bill Teahan. Text mining: A new frontier for lossless compression. In *Proceedings of the Conference on Data Compression*, DCC '99, pages 198–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0096-X. URL http://dl.acm.org/citation.cfm?id=789086.789617.

Jingwei Wu, C.W. Spitzer, A.E. Hassan, and R.C. Holt. Evolution Spectrographs: visualizing punctuated change in software evolution. In *Workshop on Principles of Software Evolution*, pages 57–66, Sept 2004. 10.1109/IWPSE.2004.1334769.

Xes-standard.org. openxes:start | xes, 2015. URL http://www.xes-standard.org/openxes/start.

Annie T. T. Ying, Gail C. Murphy, Raymond Ng, and Mark C. Chu-Carroll. Predicting Source Code Changes by Mining Change History. *IEEE Trans. Softw. Eng.*, 30(9):574–586, September 2004. 10.1109/TSE.2004.52.

Michael Zeising, Stefan Schönig, and Stefan Jablonski. Towards a Common Platform for the Support of Routine and Agile Business Processes. In *Collaborative Computing: Networking, Applications and Worksharing*, 2014.

Weidong Zhao and Xudong Zhao. Process Mining from the Organizational Perspective. In *Foundations of Intelligent Systems*, pages 701–708. 2014.

Thomas Zimmermann, Peter Weisgerber, Stephan Diehl, and Andreas Zeller. Mining Version Histories to Guide Software Changes. In *International Conference on Software Engineering (ICSE '04)*, pages 563–572. IEEE Computer Society, 2004.