MATCHING EVENTS AND ACTIVITIES

PREPROCESSING EVENT LOGS FOR PROCESS ANALYSIS

THOMAS BAIER

BUSINESS PROCESS TECHNOLOGY
HASSO PLATTNER INSTITUTE, UNIVERSITY OF POTSDAM
POTSDAM, GERMANY

DISSERTATION
ZUR ERLANGUNG DES GRADES EINES
DOKTORS DER NATURWISSENSCHAFTEN
– DR. RER. NAT. –

August 2015

Thomas Baier: Matching events and activities, Preprocessing event logs
for process analysis, © August 2015

Published online at the
Institutional Repository of the University of Potsdam:
URN urn:nbn:de:kobv:517-opus4-84548
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-84548

I like things matching.
I have an upright bass, a drum kit and a grand piano
that's the same color.

— Penn Jillette

ABSTRACT

Nowadays, business processes are increasingly supported by IT services that produce massive amounts of event data during process execution. Aiming at a better process understanding and improvement, this event data can be used to analyze processes using process mining techniques. Process models can be automatically discovered and the execution can be checked for conformance to specified behavior. Moreover, existing process models can be enhanced and annotated with valuable information, for example for performance analysis. While the maturity of process mining algorithms is increasing and more tools are entering the market, process mining projects still face the problem of different levels of abstraction when comparing events with modeled business activities. Mapping the recorded events to activities of a given process model is essential for conformance checking, annotation and understanding of process discovery results. Current approaches try to abstract from events in an automated way that does not capture the required domain knowledge to fit business activities. Such techniques can be a good way to quickly reduce complexity in process discovery. Yet, they fail to enable techniques like conformance checking or model annotation, and potentially create misleading process discovery results by not using the known business terminology.

In this thesis, we develop approaches that abstract an event log to the same level that is needed by the business. Typically, this abstraction level is defined by a given process model. Thus, the goal of this thesis is to match events from an event log to activities in a given process model. To accomplish this goal, behavioral and linguistic aspects of process models and event logs as well as domain knowledge captured in existing process documentation are taken into account to build semiautomatic matching approaches. The approaches establish a pre–processing for every available process mining technique that produces or annotates a process model, thereby reducing the manual effort for process analysts. While each of the presented approaches can be used in isolation, we also introduce a general framework for the integration of different matching approaches.

The approaches have been evaluated in case studies with industry and using a large industry process model collection and simulated event logs. The evaluation demonstrates the effectiveness and efficiency of the approaches and their robustness towards nonconforming execution logs.

ZUSAMMENFASSUNG

Heutzutage werden Geschäftsprozesse verstärkt durch IT Services unterstützt, welche große Mengen an Ereignisdaten während der Prozessausführung generieren. Mit dem Ziel eines besseren Prozessverständnisses und einer möglichen Verbesserung können diese Daten mit Hilfe von Process–Mining–Techniken analysiert werden. Prozessmodelle können dabei automatisiert erstellt werden und die Prozessausführung kann auf ihre Übereinstimmung hin geprüft werden. Weiterhin können existierende Modelle durch wertvolle Informationen erweitert und verbessert werden, beispielsweise für eine Performanceanalyse. Während der Reifegrad der Algorithmen immer weiter ansteigt, stehen Process–Mining–Projekte immer noch vor dem Problem unterschiedlicher Abstraktionsebenen von Ereignisdaten und Prozessmodellaktivitäten. Das Mapping der aufgezeichneten Ereignisse zu den Aktivitäten eines gegebenen Prozessmodells ist ein essentieller Schritt für die Übereinstimmungsanalyse, Prozessmodellerweiterungen sowie auch für das Verständnis der Modelle aus einer automatisierten Prozesserkennung. Bereits existierende Ansätze abstrahieren Ereignisse auf automatisierte Art und Weise, welche die notwendigen Domänenkenntnisse für ein Mapping zu bestehenden Geschäftsprozessaktivitäten nicht berücksichtigt. Diese Techniken können hilfreich sein, um die Komplexität eines automatisiert erstellten Prozessmodells schnell zu verringern, sie eignen sich jedoch nicht für Übereinstimmungsprüfungen oder Modellerweiterungen. Zudem können solch automatisierte Verfahren zu irreführenden Ergebnissen führen, da sie nicht die bekannte Geschäftsterminologie verwenden.

In dieser Dissertation entwickeln wir Ansätze, die ein Ereignislog auf die benötigte Abstraktionsebene bringen, welche typischerweise durch ein Prozessmodell gegeben ist. Daher ist das Ziel dieser Dissertation, die Ereignisse eines Ereignislogs den Aktivitäten eines Prozessmodells zuzuordnen. Um dieses Ziel zu erreichen, werden Verhaltens- und Sprachaspekte von Ereignislogs und Prozessmodellen sowie weitergehendes Domänenwissen einbezogen, um teilautomatisierte Zuordnungsansätze zu entwickeln. Die entwickelten Ansätze ermöglichen eine Vorverarbeitung von Ereignislogs, wodurch der notwendige manuelle Aufwand für den Einsatz von Process–Mining–Techniken verringert wird.

Die vorgestellten Ansätze wurden mit Hilfe von Industrie-Case-Studies und simulierten Ereignislogs aus einer großen Prozessmodellkollektion evaluiert. Die Ergebnisse demonstrieren die Effektivität der Ansätze und ihre Robustheit gegenüber nicht-konformem Prozessverhalten.

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

- Thomas Baier and Jan Mendling. *Bridging abstraction layers in process mining: Event to activity mapping.* In *Enterprise, Business-Process and Information Systems Modeling – 14th International Conference*, volume 147 of *Lecture Notes in Business Information Processing*, pages 109–123. Springer, 2013.
- Thomas Baier and Jan Mendling. *Bridging abstraction layers in process mining by automated matching of events and activities.* In *Business Process Management – 11th International Conference*, volume 8094 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2013.
- Thomas Baier, Jan Mendling, and Mathias Weske. *Bridging abstraction layers in process mining. Information Systems*, volume 46, pages 123–139, 2014.
- Thomas Baier, Andreas Rogge-Solti, Mathias Weske, and Jan Mendling. *Matching of events and activities – an approach based on constraint satisfaction*. In *The Practice of Enterprise Modeling – 7th IFIP WG 8.1 Working Conference*, volume 197 of *Lecture Notes in Business Information Processing*, pages 58–72. Springer, 2014.
- Thomas Baier, Andreas Rogge-Solti, Mathias Weske, and Jan Mendling. *Matching of events and activities - an approach based on behavioral constraint satisfaction.* In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1225–1230. ACM, 2015.
- Thomas Baier, Claudio Di Ciccio, Jan Mendling, and Mathias Weske. *Matching of events and activities - an approach using declarative modeling constraints.* In *Enterprise, Business-Process and Information Systems Modeling – 16th International Conference*, volume 214 of *Lecture Notes in Business Information Processing*, pages 119–134. Springer, 2015.

ACKNOWLEDGMENTS

A PhD is — what else could it be? — a process. During this process there are many people with different roles involved. First of all, I have been lucky that the role of the supervisor has been taken by Mathias Weske and Jan Mendling, who supported me in many aspects of my dissertation process. Especially their feedback on my work from sometimes very different angles was extremely helpful for me. Many thanks for this constant and great support. I am also most grateful for the discussions and feedback from Hajo Reijers, who took the role of the third reviewer of this thesis and provided valuable advice and comments for my work and gave me the possibility to stay and research at the Technical University in Eindhoven.

Other important roles in the PhD process are taken by colleagues, who contribute as co–authors, critics, discussion partners, and friends that support oneself in many ways. I have been happy to collaborate with many colleagues from different Universities. Starting with my co–authors, I like to thank Andreas Rogge–Solti and Claudio Di Ciccio for the many discussions we had and for the great input and help they provided. From my time at Humboldt University, I want to especially thank Henrik Leopold and Fabian Pittke for our fruitful discussions on label analysis and for their constant support. With my change to the Hasso Plattner Institute, I have been welcomed warmly and integrated very quickly into the group. Thanks heaps to everybody in the BPT group for the great time and the countless discussions. A special thanks goes to Marcin Hewelt for our sometimes very long debates on formal aspects of this thesis. Another important role that has been mostly taken by my colleagues, is the role of a proof–reader. I really appreciated the input and comments from the proof–reading of Kimon Batoulis, Rami Eid-Sabbagh, Henrik Leopold, Sankalita Mandal, Adriatik Nikaj, and Katrin Pieper.

Apart from the academic part of the process, there are my friends and family without whom I would not have reached this stage. Therefore, I want to express my gratitude to my parents and my brother for always supporting me in my life. Last, but by far not least, I like to thank Julia for the many ways in which she supported me, from listening over encouraging to finally also proof–reading and correcting parts of my thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

BPM	Business Process Management
BPMN	Business Process Model and Notation
CI	Configuration Item
CSP	Constraint Satisfaction Problem
DMN	Decision Model and Notation
EPC	Event-driven Process Chains
FMC	Fundamental Modeling Concepts
IT	Information Technology
ITIL	IT Infrastructure Library
LTL	Linear Temporal Logic
NLP	Natural Language Processing
OMG	Object Management Group
PAIS	Process-aware Information System
RPST	Refined Process Structure Tree
tf-idf	term frequency–inverse document frequency
UML	Unified Modeling Language

Part I

BACKGROUND

1 INTRODUCTION

With the advent of information technology (IT), companies heavily rely on IT systems for the execution of their business processes. Within the last decades, a shift from data orientation to process orientation has been observed [42, p. 5]. In this context, Business Process Management (BPM) became an important topic, as it is crucial for the design, implementation, controlling, and improvement of IT–supported business processes [143]. BPM ideally aims at a process life cycle that starts with the modeling of the processes. Next, the created model is used to implement the process in IT systems, where it is executed and monitored afterwards. From the evaluation of the execution, improvements can be made in the models and then transferred to the implementation. Yet, there is often not such a strong connection between the models and the actual implementation [116]. Sometimes there are no models at all. The research area of process mining evolved to tackle this problem by using event data from the IT systems to extract process related information [118, p. 1]. Process mining techniques are able to automatically discover process models from IT systems, check conformance of past executions with existing models, and enhance predefined models by repairing them or adding execution–related data. Many process mining case studies demonstrate the relevance and impact of these techniques in industry (see, e.g., [44, 80, 102, 125, 132, 95]).

One of the six guiding principles of the process mining manifesto [127] notes that events should be related to model elements. During the execution of a business process, many events — like the receiving of an order or the arrival of a package — occur and are automatically recorded by the supporting IT system or in some cases manually by staff in a log book. Unfortunately, these events are often not correlated to the modeled activities and thus, can hardly be used for analysis [62]. Looking at conformance checking and enhancement of process models, it is obvious that the events stemming from IT systems have to be mapped to activities defined in process models. Without a mapping from events to activities, we cannot tell which activity happened when. However, the events are typically more fine–granular than the activities defined by business users. This implies that different levels of abstraction need to be bridged in order to use the mentioned process mining techniques. Furthermore, such a mapping is not only necessary for conformance checking and process model enhancement, but also for the discovery of a process model from the supporting IT system. The benefit of a discovered process model can only be fully

exploited if the presented results are on an abstraction level that is easily understandable for the business user. However, most current process mining techniques assume that there is a one-to-one mapping between events and activities and they assume that this relation is given by simple string matching of event names and activity names. Therefore, these techniques cannot be applied without preprocessing of the event data. Available methods for event log abstraction, such as [58, 59, 75], aim to automatically provide such a preprocessing but fail to include domain knowledge in order to arrive at the required abstraction level. After all, these approaches do not try to provide a mapping to existing activities from a given process model, but try to simplify discovered models by aggregating events that seem to have a semantic relation. These techniques have limited capabilities in dealing with complex mappings between events and activities and most often neglect many-to-many relationships and concurrency in the execution. Moreover, they provide no or only limited support for correctly refining these mappings based on domain knowledge.

1.1 RESEARCH OBJECTIVE

The goal of this thesis is to provide a preprocessing of event logs that builds the connection between events in the log and activities defined by a process model. This preprocessing shall enable analysis techniques from the process mining field that rely on a binding between events and activities. Namely, these are conformance checking approaches [100, 128, 138] and process model enrichment techniques as for example decision point analysis [101], model repair [48], and performance analysis [128]. In addition to these techniques, discovery algorithms, such as [70, 123, 129, 142], can benefit from event logs that contain labels of existing model activities. By using the known terminology, the full potential of discovered process models can be leveraged and misleading results can be prevented.

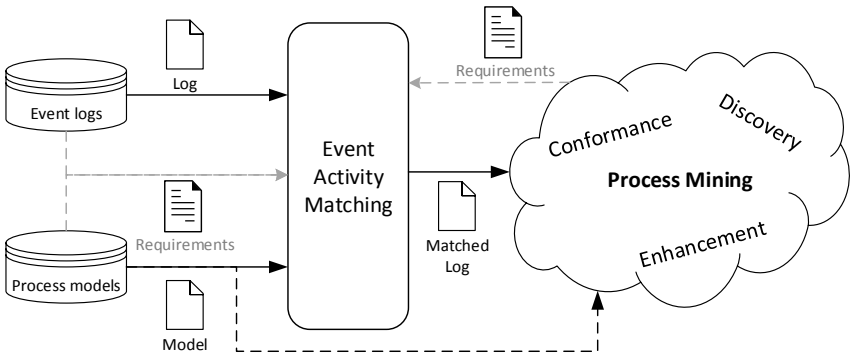


Figure 1: Overview of the matching of events and activities.

Figure 1 gives an overview of the inputs and outputs of the preprocessing approach at which we aim. The event–activity matching

takes a process model and the corresponding event log that captures the execution of the modeled process as input. The output of the matching is a preprocessed event log containing events that have a direct link to their corresponding activities from the process model. The matched event log and optionally also the process model serve as input for the various process mining techniques.

The work in this thesis follows a design science methodology as, for example, suggested by Wieringa [144]. The artifacts that are the results of this thesis are approaches for the matching of events and activities as a preprocessing for process mining techniques. In order to design such artifacts, the concrete requirements need to be elicited. The main requirements for the event–activity matching come from the process mining techniques that will consume the preprocessed event log. Yet, the event logs and process models that are used for the matching pose additional challenges in real world scenarios. Overcoming these challenges forms new requirements that originate from the structure of the input data. A rigorous requirements analysis is therefore inevitable and also part of this thesis' objective.

Having defined the requirements for the matching of events and activities, a formal understanding of the required concepts to solve matching is necessary. From this understanding, a technical solution can be built. The goal of this thesis is to create, validate and evaluate such a technical solution for the matching of events and activities and to provide automation for this solution where applicable.

1.2 CONTRIBUTIONS

In the light of the discussed research objective, we want to outline the contributions of this thesis. The main contribution of this thesis is to provide a preprocessing approach for process mining analysis that realizes the necessary binding between the activities in a process model and the events in an event log stemming from the execution of the modeled process. This contribution can be further divided into the following sub-contributions:

- *Requirements analysis.* Employing the state of the art literature as well as insights from our own case studies, we provide a rigorous analysis of the requirements for the mapping of events to their corresponding activities in a process model.

- *Formalization of the matching between events and activities.* To our best knowledge, this thesis is the first work to provide a systematic definition of the technical mapping problem for process models and their corresponding execution logs. We thereby lay the foundation not only for our own approach, but also for future work that can build on this groundwork.

- *General approach for the matching of events and activities on type and instance level.* We introduce and implement a base approach that fulfills the stated requirements. This base approach works on two levels:
  - The *type level* describes the conceptual binding of specific event types — so called event classes — to the conceptually modeled activities in a process model. Here, we provide mechanisms to manually define also complex mappings that can be used for an automated transformation of the event log.
  - The *instance level* informs about the actual instantiations of activities during the execution of a process. We assist the analyst with means to specify rules for the assignment of events to instances of activities. We introduce a tree-based clustering approach that employs the user–defined rules to transform the event log in such a way that it represents instantiations of activities from a process model.
- *Automated support of the type–level matching*. We provide automation techniques for the matching on type level for different settings:
  - *One-to-one relation* of events an activities. When events and activities are already on the same abstraction level, but their relation cannot be identified over simple name comparison, we employ replay techniques to built a constraint satisfaction problem that helps in reducing the mapping problem for the process analyst.
  - *One-to-many relation* of events and activities. For the situation where events are on a lower abstraction level and no event type belongs to multiple activities, we leverage behavioral relations in order to assist the process analyst in the mapping.
  - *Labels use natural language*. Whenever the labels from both events and activities contain natural text, we are able to employ natural language processing (NLP) techniques in order to assist in the mapping.
- *Integrated approach.* Finally, we introduce a general framework to integrate different means of automation for the type–level matching. While we demonstrate the use of this integration with the techniques developed in this thesis, the integrated approach is open to include further techniques that might be results of future work.

1.3 STRUCTURE OF THE THESIS

In order to conclude this introductory chapter, this section provides the structure of this thesis. There are three parts that shape the contents of this work:

PART I. The first part of this thesis contains this introduction as well as a chapter on preliminaries and related work. Here, the context of business process management and process mining is introduced and the formal concepts for the used process models and event logs are laid out. Moreover, the basic concepts of constraint satisfaction problems are presented. We introduce two processes that serve as running examples in the main part of this thesis. Besides an in-depth classification and discussion of related work, Chapter 2 incorporates the requirements analysis, which is one of the contributions of this thesis.

PART II. In the main part of this work, the core contributions are laid out. We start in Chapter 3 with the formalization of the matching problem and introduce the base approach, which builds the foundation for all upcoming chapters. Chapter 4, 5 and 6 provide the details for the semiautomated type–level matching. Chapter 4 introduces means to generate constraints by replaying the event log on the process model. These constraints are then used to reduce the number of possible mappings between events and activities, through which the process analyst is guided. While this replay technique only works in settings where events and activities are on the same abstraction level, Chapter 5 builds on the same idea of generating constraints, but uses more flexible behavioral relations to derive constraints. In Chapter 6, we develop an approach for the type–level matching that uses natural language text processing. Here, we also demonstrate how to incorporate further process knowledge in terms of process descriptions into the matching. Chapter 7 concludes the main part of this thesis by introducing an approach for the integration of different type–level matching techniques.

PART III. The last part of this thesis elaborates on the implementation and evaluation of the introduced concepts. In order to evaluate the different approaches we conducted different case studies with a large German IT outsourcing company and generated event logs with different characteristics from a large industry process model collection. The results outline the strengths and weaknesses of the different approaches and allow for an in-depth comparison. We conclude the thesis with an overview of the results and outline limitations and potential future work.

2 PRELIMINARIES AND RELATED WORK

This chapter provides the background of this thesis and introduces the formal concepts that are used. In Section 2.1, we introduce the BPM life–cycle and give the formal foundations for the process modeling and process mining concepts on which we ground our work. In particular, we introduce Petri nets, behavioral profiles and Declare as concepts for the representation of processes. Section 2.2 introduces the concepts of constraint satisfaction problems, which are the base for a large part of the derived approaches in this thesis. Following the introduction of preliminary concepts, two example processes, which are later used to illustrate our approaches, are introduced in Section 2.3. The chapter concludes with a classification discussion of related work and a summary of the chapter's contents.

2.1 BUSINESS PROCESS MANAGEMENT

"All work is process work" [61, p. 11]. This statement by Hammer shows the importance of business process management, as it reveals that actually every work that is being done in a company, is a process. Even business process management itself is a process. There are many different definitions of a business process [60, 121, 108, 143, 28, 43]. Weske [143, p. 5] defines a business process as "a set of activities that are performed in coordination in an organizational and technical environment". The activities are carried out to realize a specific business goal. While a business process is always executed within one organization, it potentially requires the interaction with another organization's processes.

Business process management is defined by Aalst et al. [122] as "supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information".

2.1.1 *BPM Life Cycle*

The process of business process management can be understood best by looking at the life cycle of a process described by Weske [143, p. 11ff]. Figure 2 depicts that the business process life cycle consists of four phases. The first phase is the *design and analysis* phase. In the course of this phase, business processes are identified and business process models are created to formally document, review and validate the business processes. Process models are graphical represen-

tations of a business process adhering to a given formal notation as, e.g., Business Process Model and Notation (BPMN). Once a business process is modeled, it needs to be validated and verified [43, p. 171ff]. The validation is used to check whether the process model reflects the intended or observed behavior. Thus, validation checks for semantical correctness. Verification on the other hand assures the formal correctness of a process model, i.e., whether the model complies with the rules given by the used formal modeling notation.

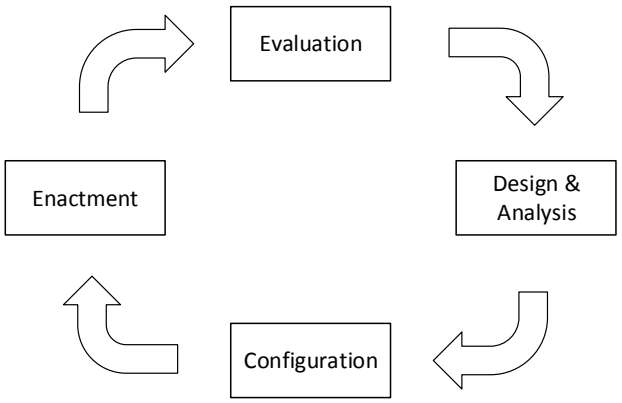


Figure 2: Business process life cycle (cf. [143, p. 12]).

Having a validated and verified process model, the next step is to implement the process in the course of the *configuration* phase. This can be done with or without the support of a dedicated IT system. If no dedicated IT system is used to implement the business process, a typical means of implementation is the use of policies and work instructions that people have to follow. Typically, these policies and work instructions are also in place when using a dedicated IT system. In order to implement a business process in a dedicated IT system, the process model needs to be enriched with more technical detail as they are typically modeled on a higher abstraction level during the design and analysis phase. The actual implementation encompasses the configuration of new and existing software systems, and the integration of different systems. The configuration phase may also include the development of new software systems. In the context of process implementation, process–aware information systems (PAIS) play an important role. Dumas et al. [42] define a PAIS as "a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models". One of the main advantages of a PAIS is that changes in the business process can be implemented without recoding parts of the IT systems. Instead, only the models need to be changed. Such changes may come from a changing business context (new competitors, new products), changes in the legal context (new laws), or from a changing technological context (new technologies) [119]. Yet, often different models are used for the initial design of the business pro-

cess than those used for implementation [116]. Therefore, changes in the business process often lead to a significant gap between business process models and implementation, even in PAIS.

After the configuration of a business process, the actual enactment is started and business process instances are executed to fulfill the goals of the organization. During this phase, data representing the current state of the process is logged by the supporting IT systems. This data can be used for instant monitoring of the actual state of the process, or for deeper process analysis in the *evaluation* phase.

The *evaluation* phase uses the data gathered during the process enactment to evaluate whether the process has been carried out according to specification, and to find potential weaknesses where the process needs to be improved. For example bottlenecks, where an activity with too few resources blocks the execution of subsequent activities, can be identified using different analysis techniques like Business Activity Monitoring (BAM) [27] or process mining [118]. The process mining research field emerged during the last decade with the goal of developing techniques to derive such information from execution data.

2.1.2 *Business Process Modeling*

Business process models are one of the main artifacts of business process management and business process modeling describes the methods to create and manage these models. Business process models are created for various purposes [43, p. 63]. For example, business process models are used to gain insight into a process and to share insights with other stakeholders. Thus, they facilitate communication about processes. Furthermore, process models can be used to configure IT systems [118, p. 8f].

A model is always an abstraction of the reality [109, p. 131]. Depending on the goal, different abstraction levels are needed. The pyramid shown in Figure 3 illustrates different levels of detail considered for process descriptions. We use the definition given by Dumas et al. [43], which contains levels 1–3 describing graphical process models, and extend it with a fourth level of textual process descriptions found in the abstraction pyramid defined by Scheer [104].

Level one starts with the process landscape model, which is the most abstract representation of business processes. Each level is linked to the next lower level (and the other way around). That is, the elements of the process landscape are connected to more concrete business processes on level two, which processes are again detailed on level three. Level three typically entails the concrete process models with precise activities, control flow, data objects, and roles. In this work, we focus on the models of this third level, as they are closest to the implementation, which is represented by the event logs used

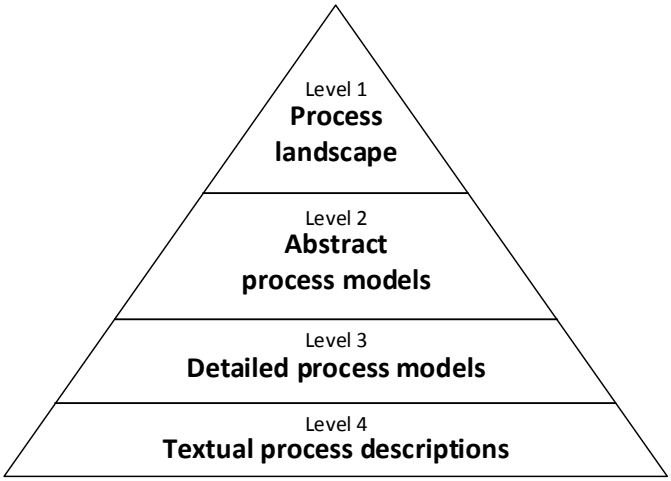


Figure 3: Different abstraction levels for process descriptions based on [43] and [104].

by the process mining techniques we want to enable. The process models and activities from level three often point to even more detailed textual descriptions found in work instructions. While these textual descriptions are most often not well structured and miss formal foundations, they can be helpful to bridge the gap between the implemented process and the process models from level three. In the course of this thesis, we show how textual descriptions can be leveraged to create the mapping between events from the implementation systems with activities from the process models.

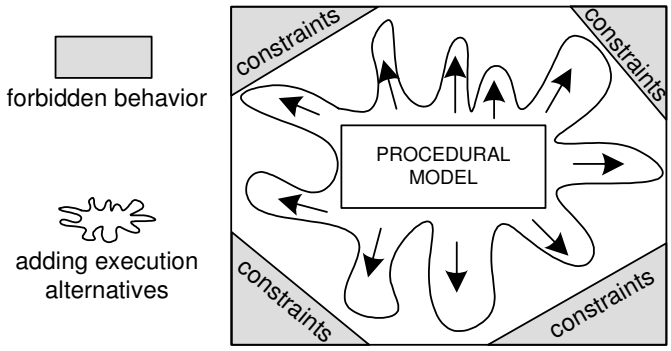


Figure 4: Imperative vs. declarative process modeling based on [87].

Turning to the models used on level three of the abstraction pyramid, there are two main categories of process models: Imperative and declarative process models [87]. The main difference between imperative and declarative modeling approaches is shown in Figure 4. Imperative models, also called procedural models, take an "inside-to-outside" approach, i.e., imperative languages explicitly define all allowed behavior. The space of allowed behavior is defined by specifying all possible execution alternatives. New alternatives have to be added explicitly to the model. In contrast to that, declarative

languages only define constraints and thereby only limit the space of allowed behavior without explicitly defining it. Thus, taking an "outside-to-inside" approach by setting limits to the space of permitted behavior using constraints. The focus of declarative modeling languages therefore lies on the *what* has to be done, while imperative languages explicitly state *how* something has to be done [50].

Imperative process modeling languages that are frequently used in practice are, e.g., Business Process Model and Notation (BPMN), Event-driven Process Chains (EPCs) [104, 29], and UML activity diagrams [83]. While these languages are widely used in industry practice, they are not well-suited for formal analysis, as they lack formal semantics and techniques for their analysis. For this reason, we use Petri nets [89] as a well-established formalism to describe procedural process models for the formal part of this thesis. For easier reading and comprehension, example processes used in this thesis are illustrated using BPMN. Most of the common modeling languages, as, e.g., BPMN and EPC, can be transformed into Petri nets [77].

For the analysis and comparison of procedural process models, behavioral profiles [138] have been proven to be a useful abstraction form. We also make use of behavioral profiles for the matching of events and activities and introduce their main concepts.

Turning to declarative process modeling, there is no widespread use of these concepts found in industrial practice. Yet, declarative process models find more and more uptake in the process mining community [96], as they are believed to yield more compact representations of very flexible processes and pertain useful properties for conformance checking [118, p. 208], [25]. There is an ongoing debate over advantages and disadvantages of declarative languages in comparison to imperative languages [50, 49, 90, 96]. The main arguments are about the understandability and maintainability of declarative models. Reijers et al. [96] argue that a combination of declarative and imperative modeling approaches would be beneficial. However, there is still not much research about the implementation of such a hybrid approach.

In this thesis, we consider imperative process models for the presentation of processes, because this is currently the de facto standard in industry. Nevertheless, we make use of the underlying concepts of the declarative modeling language Declare [126], as this has been proven beneficial for the mapping of events and activities. Therefore, we introduce both imperative and declarative modeling concepts in the next sections.

Before we turn to the actual concrete modeling languages and concepts, let us briefly define the common ground for those process models. Based on the formalization provided by Dijkman [41], we provide an abstract definition of a process model:

Definition 1 (Process model). *Let $\mathcal{L}$ be a set of labels. A process model is a tuple $P = (N, \omega, \mu)$, where*

- *N is a non-empty finite set of nodes;*
- *$\omega \subseteq N \times N$ is the set of edges, which define the control flow;*
- *$\mu = N \times \mathcal{L}$ is a function that maps nodes to labels.*

We denote $A \in N$ as the non-empty set of activities. ◇

The activities in a process model can be hierarchically related. That means an activity might be detailed using other, more fine–granular activities. These activities are referred to as sub–activities. The function $subAct : A \to \mathcal{P}(A)$ defines this hierarchical relation between activities. A process model might be further described on level four with textual descriptions entailing execution details for each activity. The function

$$desc(a) = \begin{cases} d & \text{if } subAct(a) = \emptyset \\ d \cup \bigcup_{b \in subAct(a)} desc(b) & \text{otherwise} \end{cases}$$

assigns each activity $a \in A$ the set of corresponding activity descriptions, including those linked to sub–activities. The function $desc(P) = \bigcup_{a \in A} desc(a)$ returns all activity descriptions of a process P. The next sections introduce notations for modeling processes and also include examples for the introduced formal concepts.

2.1.3 *Business Process Model and Notation*

Business Process Model and Notation (BPMN) is a standard for process modeling notation defined by the Object Management Group (OMG)[1]. In 2011, the current version BPMN 2.0 was released [56]. BPMN 2.0 includes different types of diagrams for different purposes. In this thesis, we make use of BPMN *collaboration diagrams* only. Whenever we refer to a BPMN model, a collaboration diagram is meant. The BPMN 2.0 standard introduces informal execution semantics for the provided model elements. In general, the execution semantics is inspired by Petri nets and, therefore, uses a similar means of token flow. A number of works exist that provide directions into formal semantics of BPMN and mappings from BPMN to Petri nets [77, 40, 148, 14, 134].

BPMN 2.0 provides a large set of modeling constructs and an exhaustive introduction in the complete spectrum BPMN is out of the scope of this thesis. We refer the reader to [143, 43, 105] for a comprehensive introduction into BPMN. Figure 5 shows an example process containing all constructs that are used in this thesis. Activities

1 http://www.omg.org/

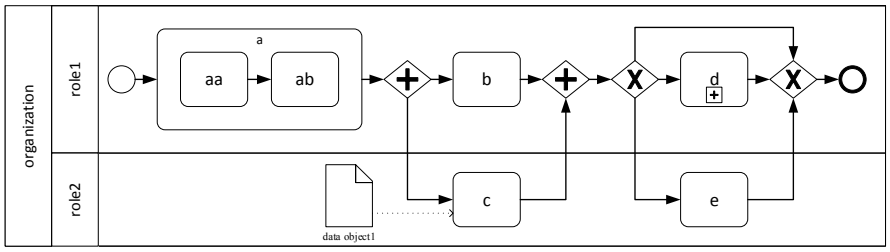


Figure 5: Example BPMN collaboration diagram.

in BPMN are modeled as rounded rectangles. BPMN distinguishes between atomic activities (sometimes called tasks) and collapsed or extended subprocesses. In Figure 5, activities aa, ab, b, c, and e are atomic activities. Activity a is an extended subprocess and activity d is a collapsed subprocess. Recalling Definition 1, the set of activities is $A = \{a, aa, ab, b, c, d, e\}$ and activity a has two sub–activities, i.e., $subAct(a) = \{aa, ab\}$. Note that the set of nodes N also includes all other objects seen in Figure 5. Yet, we do not intend to formalize these nodes in this thesis, as we make no use further use of these nodes apart from visualization.

Control flow is expressed using arcs and gateways. Gateways are represented by diamond shapes. In this thesis, only the parallel ($+$) and the exclusive ($\times$) gateway are used. In the example in Figure 5, the activities b and c are parallel to each other. In contrast, the activities d and e are exclusive to each other.

BPMN furthermore introduces events, which are represented by circles. While there is a vast amount of different events in BPMN, we only make use of start and end events. The latter are identified by a thicker line. Another aspect of BPMN is the modeling of data. Data objects can be connected to activities to show that an activity reads or writes a data object. In the given example, the data object "data object1" is read by activity c. Organizational aspects are modeled using pools and lanes. Lanes typically represent single resources, while pools group different resources. Figure 20 contains two lanes: role1 and role2. These two lanes are grouped by the pool with the label "organization".

2.1.4 *Petri Nets*

While BPMN has been chosen for the representation of process models in this thesis, Petri nets are used as the modeling language for the implementation of the approaches that are based on behavior of activities. We employ Petri nets because their semantics are well–defined and because they can be verified for correctness [114]. Petri nets are widely used in research so that there is a huge body of knowledge and analysis techniques available [81]. Most of the common modeling

languages, such as BPMN and EPC, can be transformed into Petri nets [77].

Petri nets are bipartite graphs that contain two types of nodes: places and transitions [97, p. 9]. Transitions correspond to activities and are represented by boxes. Places are depicted as circles and can hold tokens, which are indicated by black dots. In order to model the flow of tokens, i.e., the control flow, transitions and places can be connected by directed edges. Yet, it is not allowed to connect two nodes of the same type. In the following, we give the required formal definitions based on [115]. First of all, Definition 2 gives the formal definition of a Petri net.

Definition 2 (Petri net). *A Petri net is a triplet* $PN = (TR, PL, F)$*, where* TR *is the set of transitions,* PL *is the set of places, with* $TR \cap PL = \emptyset$*.* $F \subseteq (PL \times TR) \cup (TR \times PL)$ *defines the flow relation.*
Let $N = TR \cup PL$ *be the set of all nodes. The pre-set of a node* $n \in N$ *is denoted as* $\bullet n = \{m \in N \mid (m, n) \in F\}$*. The post-set of a node* n *is defined as* $n\bullet = \{m \in N \mid (n, m) \in F\}$*.* ◇

Figure 6 gives an example of a Petri net that includes the same behavior as the BPMN model shown in Figure 5. The transitions in the net either contain a label (a, b, c, d, e) or are colored black. Labelled transitions represent activities while unlabelled transitions are only necessary for syntactical reasons. Unlabelled transitions are also called *silent* transitions. We denote the set of all silent transitions as $\tau \subseteq TR$. Taking silent transitions into account, the set of transitions is split into transitions representing activities and silent transitions: $TR = A \cup \tau$.

The state of a Petri net is determined by its *marking*. The marking is defined by the function $M : PL \to \mathbb{N}_0$, which assigns a place to a natural number. $\mathbb{N}_0$ is the set of natural numbers including 0. In the graphical representation of Petri nets the marking of a Petri net is indicated by drawing black dots onto the places. The marking of the Petri net shown in Figure 6 is defined as $M(pl_1) = 1$ and $M(pl_2) = M(pl_3) = M(pl_4) = M(pl_5) = M(pl_6) = M(pl_7) = 0$. As a short representation, one can also list only the places with an indication of the number of tokens as superscript. The marking of the example Petri net is represented by the list $[pl_1^1]$.

For the execution of an activity, the corresponding transition needs to be enabled. A transition tr is enabled if there is a token on every place pl_i that is connected via an incoming arc to the transition tr, i.e., $\forall (pl_i, tr) \in F, M(pl_i) > 0$. The execution of a transition is called *firing* and consumes a token from every incoming place and produces a token on every place connected to the firing transition by an outgoing arc. Thereby, the distribution of the tokens over the places is changed. That is, a firing of a transition tr describes the change of the state (M) of a Petri net to a new state (M'), denoted as $M \xrightarrow{tr} M'$.

$M_1 \xrightarrow{*} M_n$ implies that there is a firing sequence of transitions tr_1, $tr_2, \ldots, tr_{n-1}$ such that $M_i \xrightarrow{tr_i} M_{i+1}$ for $1 \leqslant i < n$. If and only if $M \xrightarrow{*} M'$, the marking M' is called *reachable* from marking M.

There are two types of special markings of a Petri net: *initial markings* and *final markings*. An initial marking identifies the start of a process and a final marking its termination. The initial marking can be any marking where at least one transition is enabled. In the final marking there shall be only tokens in places with an empty post-set. Note that not all Petri nets necessarily contain a final marking. Nonetheless, in this thesis, we only allow Petri nets that have a given initial marking and at least one final marking.

Firing sequences starting in the initial marking M_i and leading to a final marking M_f ($M_i \xrightarrow{*} M_f$), are referred to as *sound firing sequences*. We denote the set of all sound firing sequences as SFS.

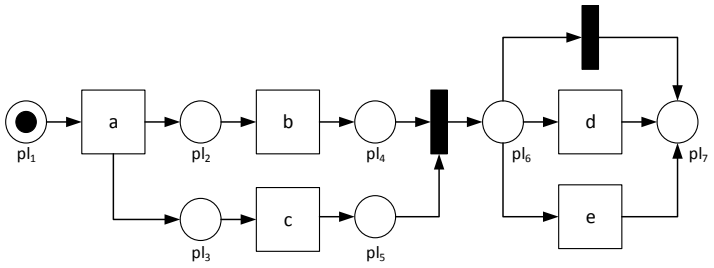


Figure 6: Petri net representation of the process shown in Figure 5.

Looking at the process illustrated in Figure 6, the initial marking is given as $M_i = [pl_1^1]$, i.e., there is only one token in the first place (pl_1), which enables transition a. Once a fires, it produces a token on place pl_2 and a token on place pl_3, which enables transitions b and c. Once b and c both have fired, there are two tokens in the places pl_4 and pl_5, enabling the connected silent transition, which immediately consumes these tokens and creates a new one on place pl_6. The transitions d and e as well as the second silent transition are now enabled. These three transitions are all exclusive to each other, because they share one input place that can only contain a single token for every execution of the process. Hence, when one of the three transitions fires, it consumes the token from place pl_6 and the other transitions are no longer enabled. After one of the three transitions has fired, a token is produced on pl_7. In this marking, no transition is enabled anymore and the process terminates. Thus, the final marking $[pl_7^1]$ is reached.

If a Petri net contains only one initial place $pl_i \in PL$ with $\bullet pl_i = \emptyset$ and one final place $pl_f \in PL$ with $pl_f \bullet = \emptyset$, and all transitions lie on a path between these two places, the net is called a workflow net [115]. The class of workflow nets has been specifically proposed for the modeling and analysis of business processes. In this thesis, we do not require process models to be transformable into workflow

nets. Nevertheless, the presence of a workflow net often allows for dedicated and more efficient analysis methods.

Another important class of Petri nets is the class of *free-choice* Petri nets [111, 13]. Free-choiceness means the separation of synchronization and conflict in Petri nets. Synchronization refers to transitions with multiple places in its pre-set and conflict relates to places with multiple transitions in its post-set. A Petri net is a free-choice net if it holds $\forall pl \in PL, tr \in TR, (pl, tr) \in F \implies \bullet tr \times pl\bullet \subseteq F$. This definition is known as extended free-choiceness as it is less restrictive as the definition that has been utilized before. Yet, it is commonly used and it has been shown that each extended free-choice net can be transformed into a free-choice net with the same behavior with respect to the labelled transitions.

Furthermore, *boundedness* is an important property of Petri nets that is referred to in this thesis. A Petri net is bounded if there exists only a finite number of markings reachable from the initial marking [114].

2.1.5 *Behavioral Profiles*

This section introduces a different, more abstract means to represent the behavior of business processes. In [136], Weidlich et al. introduce the notion of behavioral profiles. The relations defined for a behavioral profile are based on the weak order relation, denoted by $\succ$. Two activities a and b are in weak order, if a firing sequence exists where a is followed by b. Note that this does not imply that b has to directly follow a. Using the notion of weak order, three relations forming the behavioral profile are defined in Definition 3 (cf. [136]).

Definition 3 (Behavioral Profile). *Let A be the set of activities of a given process model P. Two activities $(a, b) \in A \times A$ can be in one of the three behavioral relations:*

- *The* strict order *relation $\rightsquigarrow$, if $a \succ b$ and $b \not\succ a$.*
- *The* exclusiveness *relation $+$, if $a \not\succ b$ and $b \not\succ a$.*
- *The* interleaving order *relation $\|$, if $a \succ b$ and $b \succ a$.*

The behavioral profile of A is defined as $BP_A = \{\rightsquigarrow, +, \|\}$. ◇

As a counterpart of the strict order relation, the *reverse strict* order relation is introduced. The reverse strict order relation is defined as $\rightsquigarrow^{-1} = \{(b, a) \in A \times A \mid a \rightsquigarrow b\}$. Weidlich et al. extend this behavioral profile to also include causality between activities. Therefore, the co–occurrence relation is added. Let $PN = (TR, PL, F)$ be the Petri net representation of the given process model P. Two activities $(a, b) \in TR \times TR$ are in the *co–occurrence* relation $\gg$, if for all firing sequences fs allowed by PN it holds that $a \in fs \implies b \in fs$. Table 1 and 2 show the behavioral profile and respectively the co–occurrence relation for the example process depicted in Figure 6.

Table 1: Behavioral profile of the example process in Figure 6.

	a	b	c	d	e
a	$+$	$\rightsquigarrow$	$\rightsquigarrow$	$\rightsquigarrow$	$\rightsquigarrow$
b	$\rightsquigarrow^{-1}$	$+$	$\|\|$	$\rightsquigarrow$	$\rightsquigarrow$
c	$\rightsquigarrow^{-1}$	$\|\|$	$+$	$\rightsquigarrow$	$\rightsquigarrow$
d	$\rightsquigarrow^{-1}$	$\rightsquigarrow^{-1}$	$\rightsquigarrow^{-1}$	$+$	$+$
e	$\rightsquigarrow^{-1}$	$\rightsquigarrow^{-1}$	$\rightsquigarrow^{-1}$	$+$	$+$

Table 2: Co–occurrence relation of the example process in Figure 6.

	a	b	c	d	e
a	$\gg$	$\gg$	$\gg$		
b	$\gg$	$\gg$	$\gg$		
c	$\gg$	$\gg$	$\gg$		
d	$\gg$	$\gg$	$\gg$	$\gg$	
e	$\gg$	$\gg$	$\gg$		$\gg$

Weidlich et al. present in [139] an approach to derive causal behavioral profiles from process models based on the Refined Process Structure Tree (RPST). While this approach is very efficient, it can only handle free-choice workflow nets. In [137] Weidlich et al. introduce a more generalized approach for the derivation of behavioral profiles using from bounded Petri nets. The latter approach uses Petri net unfoldings and thereby relaxes the assumption of a free-choice workflow net, but is computationally much harder than the RPST-based approach. Therefore, we use the RPST-based approach whenever possible and only fall back to the more general approach if necessary.

2.1.6 *Declare*

Having introduced the basic concepts of two imperative process modeling languages as well as the concept of behavioral profiles, we now turn to the declarative language Declare. Declare evolved from the extendable, template-based declarative language ConDec [88]. It models workflows by means of temporal rules[2].

The semantics of the used temporal rules are defined by Linear Temporal Logic (LTL) formulas. LTL is a modal temporal logic developed by Pnueli in 1977 [91]. It can be used to specify and check system properties. In order to achieve this, propositional or predicate logic is extended by modalities permitting to express the behavior of a reactive system [5]. Therefore, LTL introduces modalities referring to time that can be used to verify different properties in a linear path. Hence, the LTL language adds operators to refer to time-related properties to the basic logical operators, like $\vee$, $\wedge$, and $\neg$. Using the operator $\Diamond$ (*"eventually"*) on can specify that a property evaluates to true at at least one point in the time starting from now. The operator $\Box$ (*"always"*) expresses that a property has to hold in every state. Using the operator $\bigcirc$ (*"nexttime"*) requires a property to hold in the next

2 In literature, these temporal rules are called "constraints". In this thesis, we do not use the term "constraint" when talking about Declare in order to avoid the conflict with the term "constraints" in the context of constraint satisfaction problems (CSPs), which are introduced in Section 2.2.

state, i.e., the second state from now. With $\mathtt{prop}_1 \cup \mathtt{prop}_2$ ("$\mathtt{prop}_1$ *until* $\mathtt{prop}_2$") it is declared that a property $\mathtt{prop}_1$ has to hold until property $\mathtt{prop}_2$ holds.

Rule	Explanation	Cat.	Positive and negative examples			
Participation(a)	a occurs at least *once*	$\mathcal{C}_E$	✓ bcac	✓ bcaac	× bcc	× c
Init(a)	a is the *first* to occur	$\mathcal{C}_E$	✓ acc	✓ abac	× cc	× bac
End(a)	a is the *last* to occur	$\mathcal{C}_E$	✓ bca	✓ baca	× bc	× bac
CoExistence(a, b)	If b occurs, then a occurs, and viceversa	$\mathcal{C}_R$	✓ cacbb	✓ bcca	× cac	× bcc
NotCoExistence(a, b)	a and b never occur together	$\mathcal{C}_R$	✓ cccbbb	✓ ccac	× accbb	× bcac
Precedence(a, b)	b occurs only if preceded by a	$\mathcal{C}_R^{\rightarrow}$	✓ cacbb	✓ acc	× ccbb	× bacc
AlternatePrecedence(a, b)	Each time b occurs, it is preceded by a and no other b can recur in between	$\mathcal{C}_R^{\rightarrow}$	✓ cacba	✓ abcaacb	× cacbba	× acbb
ChainPrecedence(a, b)	Each time b occurs, then a occurs immediately beforehand	$\mathcal{C}_R^{\rightarrow}$	✓ abca	✓ abaabc	× bca	× bacb
Succession(a, b)	a occurs if and only if it is followed by b	$\mathcal{C}_R^{\rightarrow}$	✓ cacbb	✓ accb	× bac	× bcca
AlternateSuccession(a, b)	a and b if and only if the latter follows the former, and they alternate each other in the trace	$\mathcal{C}_R^{\rightarrow}$	✓ cacbab	✓ abcabc	× caacbb	× bac
ChainSuccession(a, b)	a and b occur if and only if the latter immediately follows the former	$\mathcal{C}_R^{\rightarrow}$	✓ cabab	✓ ccc	× cacb	× cbac
NotSuccession(a, b)	a can never occur before b	$\mathcal{C}_R^{\rightarrow}$	✓ bbcaa	✓ cbbca	× aacbb	× abb

Table 3: Declare templates adapted from [38].

Based on LTL, Declare rules are meant to impose specific conditions on the occurrence of tasks in execution sequences. The rationale is, that every behavior in the enactment of a process is allowed, as long as it does not violate the specified rules. That is why declarative models are said to be "open", in contrast to the "closed" fashion of classical imperative models [78]. The Declare standard provides a predefined library of templates, listing default restrictions that can be imposed on the control–flow. For a complete list of rules we refer to [120].

Table 3 encompasses the set of Declare rules that are utilized in this thesis. The table provides conforming and violating examples of execution sequences for every rule to illustrate the effects of a rule. We group the Declare rules into three different categories: existence rules, relation rules, and ordering relation rules ($\mathcal{C}_R^{\rightarrow}$).

Starting with the *existence* rules, we cluster rules that express a condition on the execution of a single activity. The set of existence rules is denoted as $\mathcal{C}_E$. For instance, *Participation*(a) is an existence rule stating that activity a must occur in every execution sequence. Further rules that fall into this category are *Init*(a) and *End*(a). These rules constrain the first and last activity of an execution sequence.

Turning to *relation* rules, we group Declare rules that constrain the execution of pairs of activities. We declare $\mathcal{C}_R$ as the set of all relation rules. *NotCoExistence*(a, b) is a relation rule that constrains a and b, and implies that a and b never occur together in the same execution sequence. In contrast to the *NotCoExistence* rule, *CoExistence*(a, b) requires that whenever activity a is part of an execution sequence, activity b is also part of the same execution sequence.

Looking at relation rules, we further differentiate rules that impose an *ordering* of activities. Templates of this category are grouped by the set $\mathcal{C}_R^{\rightarrow}$. *Precedence*(a, b) is the ordering relation rule establishing that, if b occurs in the execution sequence, then it must be *preceded* by at least one occurrence of a. *AlternatePrecedence*(a, b) is an ordering relation rule, which strengthens *Precedence*(a, b) stating that not only b must be preceded by a, but also no other b occurs between pairs of a's and b's. *ChainPrecedence*(a, b) further restricts *AlternatePrecedence*(a, b) (and *Precedence*(a, b), a fortiori), imposing that every b must be immediately preceded by a, without any other activity occurring in-between. Such transitive restrictions in rule semantics constitute the so-called *subsumption relation* among templates [36]. This relations defines that *AlternatePrecedence*(a, b) is subsumed by *Precedence*(a, b), *ChainPrecedence*(a, b) is subsumed by *AlternatePrecedence*(a, b) (and by *Precedence*(a, b)).

The subsumption relation implies that those execution sequences that are compliant with the subsumed rule, are also valid for the subsuming one. That is, execution sequences that satisfy the rule *ChainPrecedence*(a, b) also satisfy the rules *AlternatePrecedence*(a, b) and *Precedence*(a, b). From the definitions provided in Table 3, we can also state that *AlternateSuccession*(a, b) is subsumed by *Succession*(a, b), as well as *ChainSuccession*(a, b) is subsumed by *AlternateSuccession*(a, b) and *Succession*(a, b). *Succession*, *AlternateSuccession* and *ChainSuccession* rules are very similar to their corresponding precedence rules. The difference lies in the implied co–occurrence. *Precedence*(a, b) enforces that activity b can only occur if activity a has happened before, enforcing a co–occurrence of a and b only for cases that contain b. That is, activity a can occur without activity b. *Succession*(a, b) imposes the same ordering of activities a and b, but activity a cannot be executed without activity b. Thus, *Succession* subsumes *CoExistence* and adds an ordering. *AlternateSuccession* and *ChainSuccession* strengthen *Succession* in the same way as *AlternatePrecedence* and *ChainPrecedence* strengthen *Precedence*.

To exemplify the usage of the introduced Declare rules, Table 4 outlines Declare rules to closely represent the example process used in the previous sections. This time, we used the two sub–activities aa and ab instead of a. There are two important things to note. First, the behavior captured by the rules given in Table 4 do not exactly reflect the behavior of the process given in the BPMN model in Figure 5. For example, the activities d and e are only restrained in that they are exclusive to each other and that they cannot happen before any of the other activities. Yet, both d and e could be executed multiple times without violating any of the given rules. The second important thing to note is that there are obviously multiple ways to model the same behavior with other rules. Hence, Table 4 does not give the only possible way of modeling the example process with Declare rules.

Table 4: Exemplary Declare rules for the process shown in Figure 5.

Declare rule	Explanation
Init(aa)	The process starts with aa.
ChainSuccession(aa, ab)	aa and ab only occur iff aa occurs directly before ab.
Succession(ab, b)	ab only occurs iff followed by b.
Succession(ab, c)	ab only occurs iff followed by c.
Precedence(b, d)	d occurs only if preceded by b.
Precedence(c, d)	d occurs only if preceded by c.
Precedence(b, e)	e occurs only if preceded by b.
Precedence(c, e)	e occurs only if preceded by c.
NotCoExistence(d, e)	d and e never occur together.

2.1.7 *Process Execution*

Once a business process is designed and configured, it is enacted and thus, executed. The execution can be performed completely automatically, completely manually, or a mixture of both. In this thesis, we do not differentiate between automatic or manual executions, yet, we will impose certain restrictions on how the execution of a process has to be logged in order to apply the introduced techniques. This is covered in the next section, which deals with process mining. In this section, the formal foundations are laid to cover business process execution.

When a process is executed, the activities of the corresponding process model are instantiated. During the life time of an activity instance, the activity instance is in different states. There are different life cycle models proposed in literature (see, e.g., [118, p. 101], [143, p.

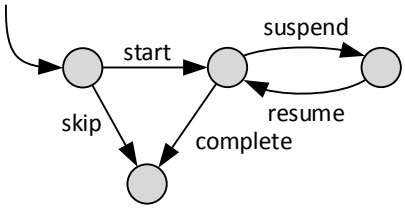


Figure 7: Activity life cycle model based on [118, p. 101].

83ff.]). In this thesis, we adopt the life cycle model proposed by Aalst in [118, p. 101]. Figure 7 shows the activity life cycle model based on the model proposed by Aalst. Definition 4 provides the formal definition of an activity life cycle model.

Definition 4 (Activity life cycle model). *Let* LCS *be a set of states, and* LT *be the set of activity life cycle transactions. An activity life cycle model* $\mathrm{ALM} = (\mathrm{LCS}, \mathrm{lcs}_\mathrm{I}, \mathrm{lcs}_\mathrm{F}, \mathrm{LT}, \theta)$ *is a transition system that defines the allowed sequences of life cycle transactions. Here,* $\theta \subseteq \mathrm{LCS} \times \mathrm{LT} \times \mathrm{LCS}$ *is a finite set of transition relations modeling the allowed life cycle transitions in a given state. An activity life cycle model has an initial state* $\mathrm{lcs}_\mathrm{I} \in \mathrm{LCS}$ *and a final state* $\mathrm{lcs}_\mathrm{F} \in \mathrm{LCS}$. ◇

As shown in Figure 7, we use a reduced set of activity life cycle transitions $\mathrm{LT} \subseteq \{\text{start, skip, suspend, resume, complete}\}$. There are two reasons for reducing the set of transitions proposed by Aalst. The first reason is simplification. In order to introduce the concepts developed in this thesis, an excessive set of life cycle transitions is not necessary. Moreover, the concepts do not heavily depend on the specific transitions and the set can therefore easily be altered and extended if necessary. The second reason, why we chose the specific set of life cycle transitions is that these are the transitions we encountered in our industry case studies.

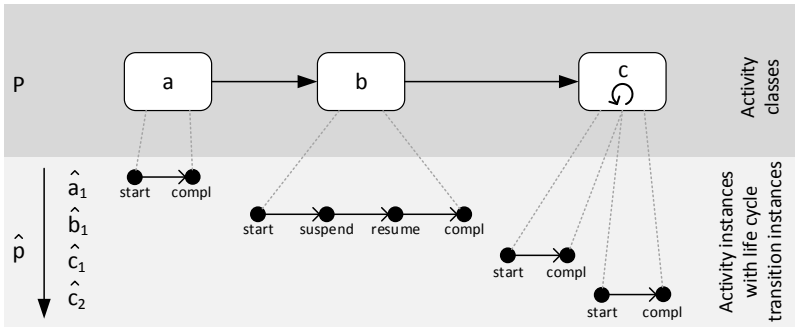


Figure 8: Process model and process instance with event diagram of life cycle transitions.

For the instantiation of activity life cycle models, event diagrams are used as proposed by Weske [143, 86f.]. Figure 8 shows a simple example process P and a corresponding process instance $\hat{p}$ with the events for each life cycle transition of an activity. Events are repre-

sented by black bullets and represent the execution of a life cycle transition. Time proceeds from left to right in an event diagram.

Figure 9 shows the relations of model and instance level. When a process is executed, this is interpreted as an instantiation of the process model, which can happen zero or many times. A process model has one or more activities that get instantiated during process execution. Activities can have sub–activities, thus there is a relation between activities as well as between activity instances. Each activity has a life cycle model and the instantiation of this life cycle model is connected to the corresponding activity instance. On the lowest level, the instances of the life cycle transitions are connected to their corresponding life cycle instance.

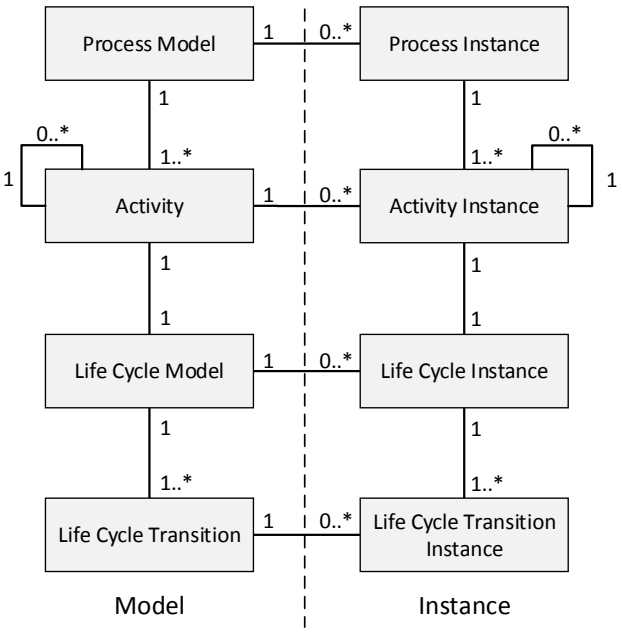


Figure 9: Model and instance level of processes.

2.1.8 *Process Mining*

2.1.8.1 *Overview of Process Mining Techniques*

Process mining refers to a toolset of technologies to leverage the information recorded by IT systems that support business processes. Automated discovery of information from event logs created by IT systems is the main goal of process mining [113]. Figure 10 gives an overview of the general process mining setting. Processes are carried out by machines and people with the help of software systems. These software systems record event logs, which range from simple text files to data hidden in complex structures of large databases. Once we have extracted an event log from the IT systems, process min-

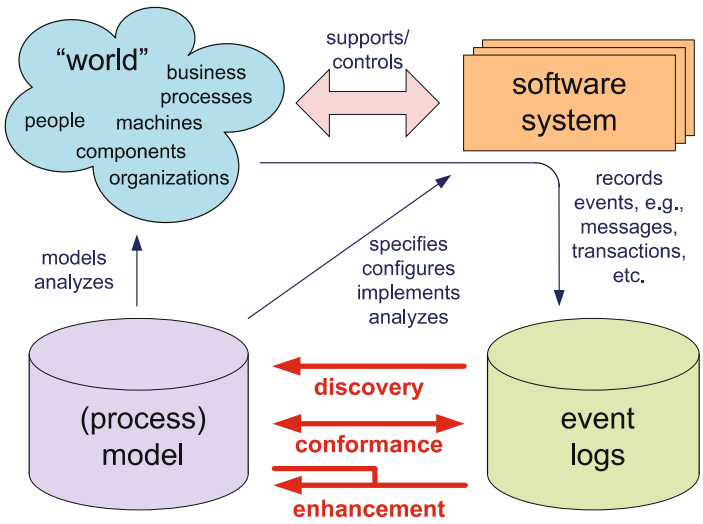


Figure 10: Overview of process mining [118, p. 9].

ing provides three types of actions: Discovery, conformance checking, and enhancement.

There is a plethora of different *discovery* algorithms, which distinguish three different perspectives: the process, the organization or the case [130]. The aim of the discovery algorithms that focus on the process perspective is to generate a process model covering the behavior found in the event log. In this area currently the majority of discovery algorithms can be found and there are many different approaches such as the α–algorithm and its extensions [123, 31, 74], heuristic mining [142], fuzzy mining [58], genetic mining [32, 22], region–based mining [129], and inductive mining [70]. The organizational perspective deals for example with the relations between persons and roles based on their handover of work or based on the execution of similar activities. The case perspective focuses on the different characterizations of a process instance. It analyzes the data elements as well as the roles and persons contributing to a process instance.

Turning to the second process mining technique, the *conformance checking*, we deal with the comparison of the process as recorded in an event log to a given process definition, e.g., a process model. Different approaches have been suggested as means for conformance checking. Rozinat et al. [100] introduce a conformance checker plug–in in ProM 5, which employs a log replay technique to check to which degree an event log matches the behavior specified in a given Petri net. Another technique is provided by Aalst et al. [128], who define alignments of event log and process model to pinpoint deviations. Based on the behavioral profile described in Section 2.1.5, Weidlich et al. [138] suggest a conformance checking approach that is less strict than the previous ones. The majority of approaches focuses on conformance of event logs to process models. In contrast to this, van der Aalst et al. [124] employ LTL formulas to specify business rules and

to check these against the process execution recorded in an event log. While no process model is necessary for this approach, the activities found in process models are typically used in the specified business rules.

The third technique in process mining is called *enrichment* and deals with the improvement of process models based on the information captured in event logs [118, p. 10]. For example, one can extract performance data as, such as activity durations or waiting times, and plot this information onto process models. Another example of enrichment is decision point analysis introduced by Rozinat and Aalst in [101]. Here, the event log is analyzed to extract rules describing the reasons for choosing between exclusive paths in a process model.

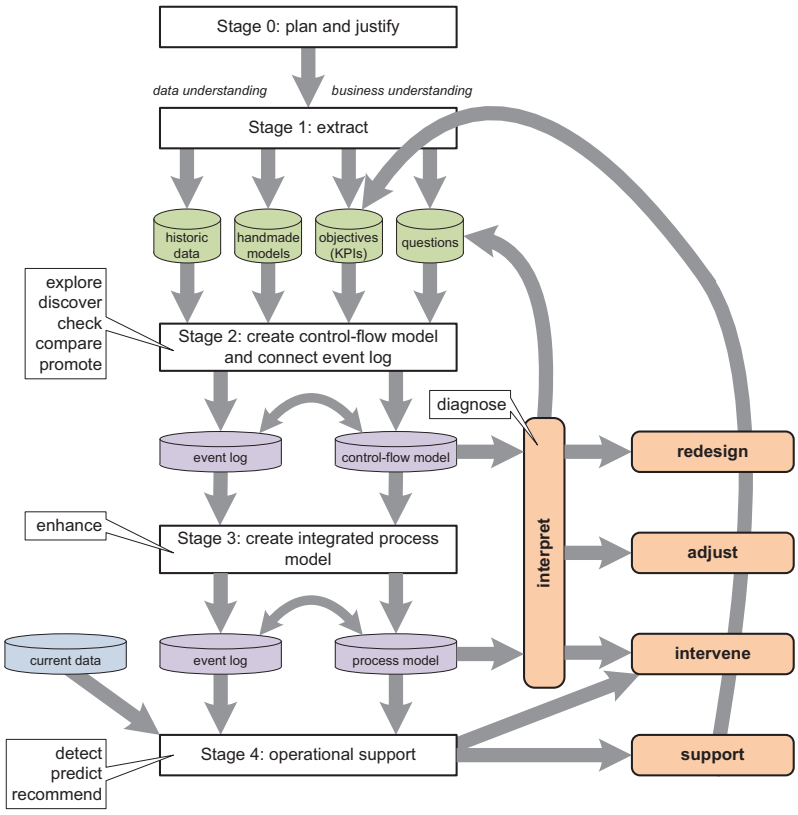


Figure 11: The L* life cycle model for process mining projects [117].

There are different approaches towards structuring a process mining project, such as [117, 19, 132]. Figure 11 shows the L* life cycle of a process mining project as introduced by Aalst [117]. The life cycle starts with the planning and justification of the project. Having a concrete plan, the required data is gathered in Stage 1. Besides event logs, also handmade models are collected for conformance or enhancement activities. Furthermore, objectives should be set and questions for which an answer is sought, should be formulated. Stage 2 aims at building a control–flow model that is tightly connected to the

given event log. This can be done by discovering a process model from the event log, by verifying an existing handmade model using conformance checking, or by merging an existing handmade model with a discovered model. Having a process model that reasonably represents the recorded executions, previously formulated questions might already be answered and actions like redesign can be initiated. Next, an integrated process model is built using enhancement techniques in Stage 3. The integrated model can again be used to answer questions and take actions. In Stage 4 the integrated model is used in conjunction with current data of the IT systems, to provide operational support. For example, the duration of currently running process instances can be predicted, or activities can be recommended to process participants on the basis of the historic data encapsulated in the event log and integrated process model.

2.1.8.2 *Fundamental Concepts of Process Mining*

Coming from the main applications of process mining, this section introduces the fundamental formal background on which we will ground the work in the upcoming chapters. The basis for every process mining algorithm are events recorded by one or more IT systems during the execution of a business process. Etzion and Niblett [45, p.4] define an event as follows:

> "An event is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. The word event is also used to mean a programming entity that represents such an occurrence in a computing system."

Etzion and Niblett differentiate between "something that has happened", i.e., a real world event as, e.g., an incoming call in a help desk, and the "programming entity" that reflects the occurrence of the real world event in an IT system. The authors also stress that a single occurrence of an event can be represented by multiple event entities. These entities might be stored in one or in multiple distributed systems.

In this thesis, we refer to an event occurrence as *event instance* of a particular *event class*. Whenever the term event is used in this thesis, it is used as a synonym for event instance. Definition 5 introduces the formal terms for events, which are used throughout this thesis.

Definition 5 (Event class, event instance, event attributes). *Let* $\hat{E}$ *be the set of event instances. Each event instance has different attributes, which are specified in the set of all attributes, called ATE. For each attribute* $\#_{attr}(\hat{e})$ *refers to the value of the attribute* $attr \in ATE$ *for the event* $\hat{e} \in \hat{E}$. *An event instance has at least the attribute* $\#_{\mathtt{time}}(\hat{e})$, *which refers to the time when it occurred, and the attribute* $\#_{\mathtt{class}}(\hat{e}) = e$, *which links to the event class* $e \in E$, *where* E *is the non-empty set of event classes.* $\diamond$

As stated in Definition 5, an event instance is a particular occurrence of an event belonging to a specific event class. For example, the incoming call from Mrs. Smith at 4.35 PM on July 23, 2014 is an event instance of the event class "Incoming call". Formally, there is an event class, $e =$ "Incoming call", and an event instance, $\hat{e}$, with the attributes $\#_{time}(\hat{e}) =$ "07/23/2014 16:35:00" and $\#_{class}(\hat{e}) = e$. We use the circumflex over the symbol of an event class to imply an event instance of this particular class. That is, $\hat{e}$ implies that $\#_{class}(\hat{e}) = e$. Note that the described attributes are only the mandatory attributes we assume for every event. Other attributes may contain role information or values of changed database fields. For the example event there might be an attribute capturing the phone number of the calling person, e.g., $\#_{phone}(\hat{e}) =$ "+493012345678".

In order to gain insight into a particular process, event instances need to be correlated to their corresponding process instance. Event instances belonging to the same process instance are grouped in a trace. An event log groups a set of traces for a single process.

Definition 6 (Trace, event log). *A trace is a list of the form $\hat{E}^*$, where the contained event instances are ordered by their time attribute. The ordering of two event instances is furthermore captured in the ordering relation $>^{\hat{e}} \in \hat{E} \times \hat{E}$. We write $\hat{e}_1 >^{\hat{e}} \hat{e}_2$ to indicate that event instance $\hat{e}_1$ occurs before event instance $\hat{e}_2$ in a trace. A trace can have attributes assigned to it. The set of all possible trace attributes is called ATT and $\#_{attr}(t)$ refers to the value of a trace attribute $attr \in ATT$. An event log L is a set of traces.* ◇

Like events, traces have attributes. Regarding trace attributes, we only assume the attribute $\#_{case}(t)$, which uniquely relates a trace to a process instance, as mandatory. Nevertheless, more attributes can be defined and used in the matching of events and activities.

Every event instance is assigned to a trace and each trace in an event log is related to a process instance. Note that the relation of event instances to traces might not be trivial in every practical setting. Yet, there is plenty of work on event correlation that identifies the relation between event instances and traces (see, e.g., [86, 103]). In this work, we therefore assume this relation to be already given. Although each trace can contain multiple instances of the same event class, we assume that each event instance in a trace can be uniquely identified by its attribute values.

Let $E = \{k, l, m\}$ with $k =$ "Incoming call", $l =$ "Ticket created" and $m =$ "Ticket closed" be the set of event classes. Consider the trace $t_1 = \langle \hat{k}_1, \hat{l}_1, \hat{k}_2, \hat{m}_1 \rangle$. We use indices to be able to directly refer to event instances from event classes that occur more than once. Yet, there are multiple ways of describing traces for instances of these event classes. In case we do not need to make direct reference to single event instances, we may omit the indices. In this case, we can also write $t_1 = \langle$Incoming call, Ticket created, Incoming call, Ticket closed$\rangle$.

Traces might be split for analysis of particular parts of a trace. A continuous part of a trace is called a sub-trace. We use the | operator for the concatenation of traces or sub-traces. Looking at trace t_1, we could declare $t_{11} = \langle \hat{k}_1, \hat{l}_1 \rangle$ and $t_{12} = \langle \hat{k}_2, \hat{m}_1 \rangle$. Then, $t_1 = t_{11}|t_{12}$.

Typically, there are many traces in an event log that contain the exact same ordering of events. Consider the event log $L_1 = \{t_1 = \langle \hat{k}, \hat{l}, \hat{m} \rangle, t_2 = \langle \hat{k}, \hat{m}, \hat{l} \rangle, t_3 = \langle \hat{k}, \hat{l}, \hat{m} \rangle\}$. The traces t_1 and t_3 are identical, if we abstract from event attributes and regard traces as lists of event class labels. The specific ordering of events is referred to as a variant, denoted as $v \in V$, where V is the list of all variants. Using variants, the event log L_1 can also be written as $L_1 = \{v_1 = \langle k, l, m \rangle^2, v_2 = \langle k, m, l \rangle\}$. The number of occurrences of a variant is indicated as a superscript. If a variant is represented by only one trace, the superscript is omitted. As a generalization, we also write $V = \langle v_1^{w_1}, \ldots, v_i^{w_i} \rangle$, where the number of occurrences of a variant, called weight, is represented by w. The list of all variant weights is $W = \langle w_1, \ldots, w_i \rangle$. The weight at index i in W refers to the variant at index i in V.

2.2 CONSTRAINT SATISFACTION PROBLEM SOLVING

This section introduces the general concepts of constraint satisfaction problems, which will play a major role in the matching of events and activities in this thesis. Constraint programming makes use of a combination of reasoning and computing techniques. It is applied in a wide range of disciplines from molecular biology over electrical engineering to business applications like option trading. In order to apply constraint programming to a real world problem, it first needs to be transformed into a constraint satisfaction problem (CSP). This transformation is done in two steps: First, variables are introduced that range over specific domains. Second, constraints over these variables are defined. Such constraints are usually a subset of first-order logic. The definition of a constraint satisfaction problem is called modeling [2, p. 1ff].

Formally, a CSP is a triple $CSP = (\mathcal{X}, \mathcal{D}, CST)$ where $\mathcal{X} = \langle x_1, x_2, \ldots, x_n \rangle$ is an n-tuple of variables with the corresponding domains specified in the n-tuple $\mathcal{D} = \langle d_1, d_2, \ldots, d_n \rangle$ such that $x_i \in d_i$, with $i \in 1..n$ [53]. $CST = \langle cst_1, cst_2, \ldots, cst_t \rangle$ is the t-tuple of constraints. We use predicate logic to express the constraints used in this paper.

Once the real world problem is expressed in terms of a CSP, the CSP needs to be solved. The solution of a CSP can be approached using either domain specific methods like, e.g., linear programming, or general methods, which employ specific search methods to reduce the search space [2, p. 2]. Also a combination of domain specific and general methods is possible.

The set of solutions to a CSP is formally denoted as $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m\}$, where each solution $\mathcal{S}_k = \langle s_1, s_2, \ldots, s_n \rangle$ is an n-tuple with

$k \in 1..m$, $s_i \in d_i, i \in 1..n$, and every constraint in CST is satisfied. The value s_i at index i in a solution $\mathcal{S}_k$ represents the value of the variable x_i at index i in $\mathcal{X}$.

To exemplify the use of a CSP, consider the following classic example taken from [2]. The equation SEND + MORE = MONEY is given and the task is to replace each letter by a different digit in such a way that the equation is correct. In order to express this as a CSP, the variables are defined as $\mathcal{X} = \langle S, E, N, D, M, O, R, Y \rangle$. The domain of each of the variables is the integer interval $[0, 9]$ except for variables S and M, which have the interval of $[1, 9]$ as their domain since they are the leading digits. One constraint $cst_1 \in CST$ is defined as follows.

$$\begin{aligned} cst_1 \equiv \quad & 1000 \cdot S + 100 \cdot E + 10 \cdot N + D \\ + \; & 1000 \cdot M + 100 \cdot O + 10 \cdot R + E \\ = \; & 10000 \cdot M + 1000 \cdot O + 100 \cdot N + 10 \cdot E + Y \end{aligned} \tag{1}$$

There is exactly one solution to this CSP, which leads to the correct equation $9567 + 1085 = 10652$. In order to solve such a CSP, different algorithms exist that fall into one of three categories: inference, search, or a combination of both [53]. In this thesis, we will not go into the details of algorithms for solving CSPs but rely on existing implementations that automatically detect an appropriate search strategy.

2.3 ILLUSTRATING EXAMPLES

This section introduces the two example processes that we will use to illustrate both the requirements that need to be fulfilled by the methods we introduce, as well as the methods and concepts themselves. The two example processes demonstrate different features of IT supported business processes found in real life scenarios.

2.3.1 *An Order Process*

Starting with an example of an order process, Figure 12 depicts a process model that starts with the activity "Check order", specifies the handling of an order, and ends with the archiving of the processed order. Table 5 shows an exemplary event log with 5 traces, which have been produced by an IT system supporting the order process depicted in Figure 12. Obviously, it is not straightforward to interpret the given event log, because the event labels are cryptic database field names, which cannot be easily matched to the names of the activities in the process model. Yet, once the mapping is established as shown

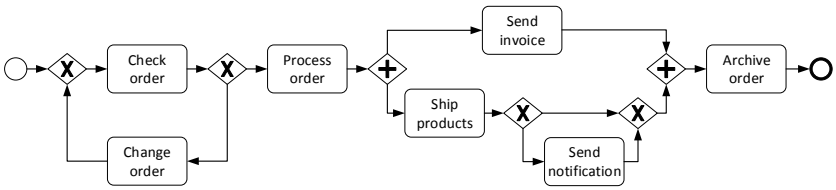


Figure 12: Order process model in BPMN.

in Table 6, we can use the event log to check conformance between the model and the log. For example, we are able to detect that in the case represented by trace t_3, the customer has already been notified before the products were shipped. According to the process model, this should not happen. It is critical for organizations to detect, and accordingly react to such nonconforming behavior [118].

Table 5: Event log L_1 of order process P_{Ord}.

	Label sequence
t_1	⟨ O_CHK, O_PRC, I_SM, P_SP, O_ARC ⟩
t_2	⟨ O_CHK, O_RCO, O_CHK, O_PRC, P_SP, P_NOT, I_SM O_ARC ⟩
t_3	⟨ O_CHK, O_PRC, I_SM, P_NOT, P_SP, O_ARC ⟩
t_4	⟨ O_CHK, O_PRC, O_RCO, P_SP, P_NOT, I_SM, O_ARC ⟩
t_5	⟨ O_CHK, O_PRC, P_SP, I_SM, P_NOT, O_ARC ⟩

Table 6: Mapping for L_1 and P_{Ord}.

Activity	Event class
Check order	O_CHK
Change order	O_RCO
Process order	O_PRC
Send invoice	I_SM
Ship Products	P_SP
Send notification	P_NOT
Archive order	O_ARC

Table 7: Event log L_2 of order process P_{Ord}.

	Label sequence
t_1	⟨ O_CHK_S, O_PR_S, O_PR_E, I_SM_E, P_SP_E, O_ARC_S, O_ARC_E ⟩
t_2	⟨ O_CHK_S, O_RC_SB, O_RC_E, O_CHK_S, O_PR_S, O_PR_E, P_SP_E, P_NOT_E, I_SM_E O_ARC_S, O_ARC_E ⟩
t_3	⟨ O_CHK_S, O_RC_SA, O_RC_E, O_CHK_S, O_PR_S, O_PR_E, P_NOT_E, I_SM_E, P_SP_E, O_ARC_S, O_ARC_E ⟩
t_4	⟨ O_CHK_S, O_PR_S, O_PR_E, P_SP_E, P_NOT_E, I_SM_E, O_ARC_S, O_ARC_E ⟩
t_5	⟨ O_CHK_S, O_PR_S, O_PR_E, P_SP_E, P_NOT_E, I_SM_E, O_ARC_S, O_ARC_E ⟩

Table 8: Mapping for L_2 and P_{Ord}.

Activity	Event class
Check order	O_CHK_S
Change order	O_RC_SA, O_RC_SB, O_RC_E
Process order	O_PR_S, O_PR_E
Send invoice	I_SM_E
Ship products	P_SP_E
Send notification	P_NOT_E
Archive order	O_ARC_S, O_ARC_E

Another example of an event log (L_2) from an IT system that supports the order process is given in Table 6. Yet, this time, more events have been logged by the system. By simply counting the number of different event classes in the log, it can be seen that there are more event classes than activities in the process model. Table 8 provides the required mapping between events and activities. For some of the activities multiple events are being logged, while for others only one type of events can be observed. Looking at the events of the "Change order" activity, not necessarily all events belonging to an activity are generated when the activity is executed. The event instances of classes O_RC_SA and O_RC_SB are exclusive to each other, i.e., they never occur together in a single trace. In contrast to this, the events belonging to activity "Process order" and "Archive order" always occur together in a trace. Having the mapping provided in Table 8, one can again conduct a conformance analysis to detect that in trace t_3 the notification has been sent out before the products were shipped.

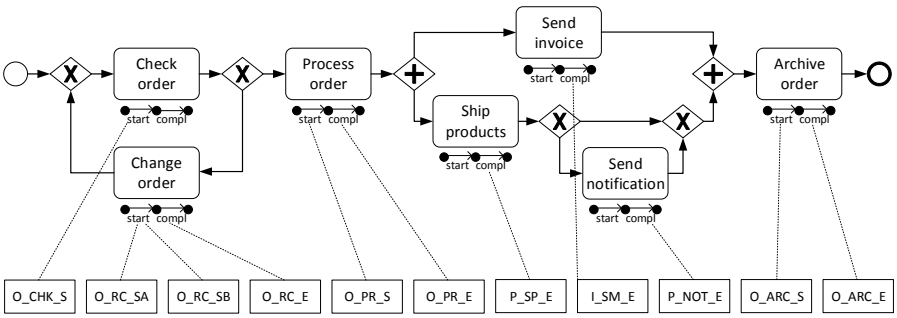


Figure 13: Order process with life cycle transitions and their mappings to event classes.

Figure 13 sketches another type of mapping. In addition to the typical process model, each activity is annotated with its life cycle. Here, the life cycle for each activity is the same, consisting of a start

and an end transition. The dotted lines represent a mapping of event classes to the activity life cycle transitions. While some of the life cycle transitions cannot be mapped to any event class, the start transition of "Change order" has two different event classes assigned to it. For those activities, where both start and complete transitions are mapped to event classes, it is now possible to compute the run time of the activity instances for each process instance. These activity durations can then be plotted onto the process model, as many popular process mining tools as, e.g., Disco[3] and Celonis Process Mining[4] do.

2.3.2 *An Incident Management Process*

In this section, we introduce another example process that is used for the illustration of our concepts and the requirements imposed on them. Figure 14 depicts the process of incident management based on the definition found in the IT Infrastructure Library (ITIL) [23]. The process is executed by three different roles. The main role is the first level, which is responsible for logging, classifying and initial diagnosis of an incident. In case a first level agent cannot resolve the incident on their own, the incident can be functionally escalated to one of the other two roles, i.e., a second level agent or to security. In any case, the first level performs the final resolution and recovery and closes the incident. Table 9 entails further descriptions of the activi-

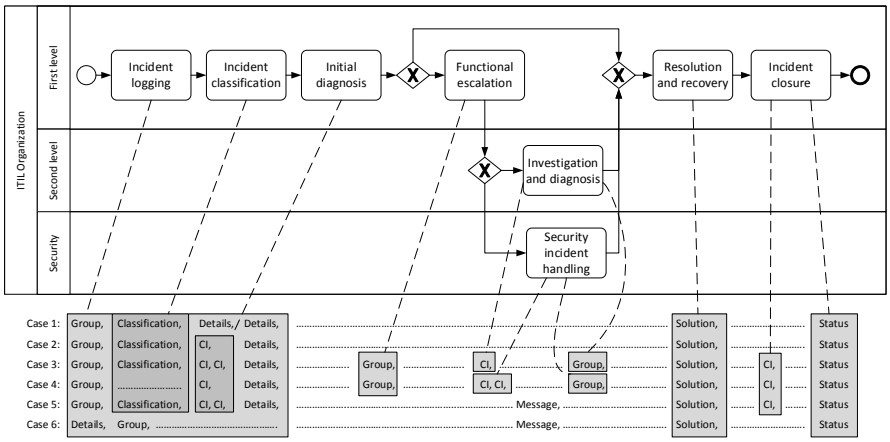


Figure 14: Example of event to activity relations: Incident Management process model and low-level event log with shared functionalities and concurrency

ties contained in the process model in Figure 14. Such descriptions are often attached in process modeling tools or separately provided in more detailed work instructions. The goal of these descriptions is to give a better understanding of the tasks that need be carried out in order to perform a given activity. While our exemplifying descrip-

3 See http://fluxicon.com/disco/
4 See http://www.celonis.de/en/discover/our-product

tions are rather short, these textual instructions can be very long and comprehensive in practical settings.

Table 9: Activity descriptions for the incident process

Activity	Description
Incident logging	The responsible group within the 1st level is automatically assigned. A member of this group needs to log the details of the incident.
Incident classification	Depending on the logged details, the appropriate classification needs to be chosen.
Initial diagnosis	The assigned 1st level supporter needs to research the knowledge base for the described problem and has to detect the configuration item (CI) that needs fixing.
Functional escalation	If no solution can be found in the knowledge base, the 1st level supporter has to route the incident ticket to the responsible 2nd level group or to the security team.
Investigation and diagnosis	A 2nd level supporter needs to perform a technical investigation and diagnosis of the reported incident, select the correct configuration item (CI), and report the solution back to the 1st level group.
Security incident handling	The security group needs to track a potential attack of an IT system in the security database and has to perform all required actions to resolve the issue. The solution is then passed back to the responsible 1st level group.
Resolution and recovery	Once the solution for the incident is found, it needs to be logged and, if necessary, saved to the knowledge base. If required, the customer is informed.
Incident closure	Finally, the configuration item (CI) and documentation of the incident is checked and the incident is closed.

Besides the process model, Figure 14 depicts an excerpt of an event log with six traces. In contrast to the example logs shown for the order process in the previous section, the events recorded by the supporting IT system for the incident management process contain actual human readable words. Still, the relation between events and activities cannot be easily done using simple string matching, as the terms used in the event log only rarely occur in the names of the activities. For instance, the two event classes “Group” and “Details” have to be related to the activity “Incident logging”. Again, there are sometimes multiple event classes assigned to some of the activities. These may be related to life cycle transitions of those activities and thereby enable performance analysis for these activities in the same way as shown for the order process. What is more interesting in this example, is the fact that there are event classes that are mapped to multiple activities. For example, event instances of the event class “CI” can be assigned to four different activities. The activity “Initial diagnosis” can be reflected by an event of the class “CI”, but also the activities “Investigation and Diagnosis”, “Security incident handling”, or to the activity “Incident closure” can be mapped to such an event. Similarly,

the events of class "Group" are mapped to either one of the activities "Incident logging", "Functional escalation" "Investigation and Diagnosis", or "Security incident handling". Thus, these events represent some common functionality that can be used by multiple activities and which always logs the same type of events.

Furthermore, there are events of the event class "Message", which are not related to any activity. These may stem from system activities that are not actually part of the incident process but still related to the case. Such activities could for instance include a monitoring of the incident cases.

It can also be seen that the execution is almost never according to the process model. The process model requires a sequential execution of the activities "Incident logging", "Incident classification" and "Initial diagnosis". Yet, in the event log there are events related to the activity "Incident logging" before and after the events recorded for the other two activities. Thus, it seems that the activity "Incident logging" is running in parallel to "Incident classification" and "Initial diagnosis".

In contrast to the non-conforming behavior seen in the order process, the incident process does not contain only rare non-conforming execution but rather systematically differs from the designed process model in some of its parts. A matching approach has to take such circumstances into consideration. The next section will elaborate on the concrete requirements for the matching approach in more detail.

2.4 REQUIREMENTS

The goal of this thesis is to provide a preprocessing of event logs used by different process analysis techniques, such as process discovery, conformance, and enhancement techniques. Based on this goal, this section defines the requirements such a preprocessing approach needs to fulfill. The requirements that we present have been assembled based on existing literature as well as on the data of the real life case studies that we have conducted.

Starting with techniques that measure the conformance of an event log to a given process model, it can easily be seen that activities and events have to be matched in order to know which activity in the model has been executed when a particular event is seen in the event log. Currently available approaches based on log replay (e.g., [100]) or alignment (e.g., [128]) assume that events and activities can be matched over their labels, thereby implying a one-to-one relation between activities and events on both type and instance level. Furthermore, most of the process discovery techniques (e.g., [70, 123, 129, 142]) also use the label of an event as the label of the corresponding activity in a discovered process model. In order to make a discovered process model easily understandable for business analysts, the

model activities should use business activity names instead of event names. Thus, also for process discovery it is required to match events to activities.

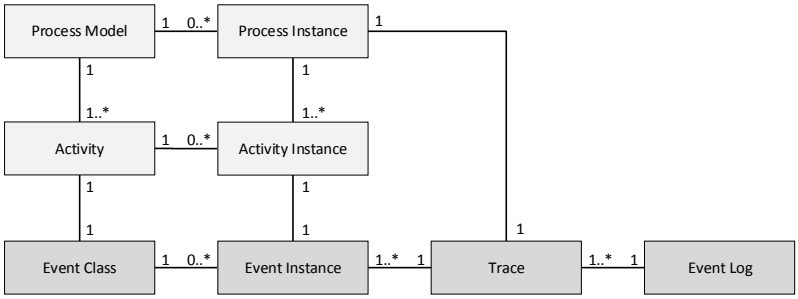


Figure 15: Relations of the entities of the event log, process model, and process execution with assumed cardinalities

Figure 15 shows the relation between process models and event logs and cardinalities assumed by most conformance checking and discovery techniques. A process model is connected to process instances reflecting the actual executions of the modeled process as explained in Section 2.1.7. It is assumed that there is a trace in the event log for each process instance and that each event instance is related to exactly one trace. Note that the relation of event instances to traces might not be trivial in every practical setting. Yet, there is plenty of work on event correlation, which relates event instances to traces (see, e.g., [86, 103]). In this work we therefore assume that this relation is already given. Furthermore, we assume that the link between a process instance and a trace is given. As process models represent the design of a process and event logs represent the execution of a process, activities in a process model cannot be directly linked to event instances in an event log. Instead, event instances need to be linked to the execution instances of activities, called activity instances.

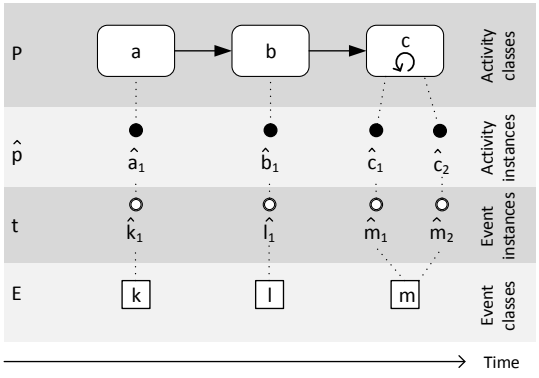


Figure 16: Event classes and instances matched in a one-to-one relation to activities and activity instances

A small example of a mapping with a one-to-one relation on both type and instance level is depicted in Figure 16. Figure 16 shows a

simple process P with a sequence of the three activities a, b, and c, with c being an activity that can be repeated in a loop. On the level below the process model, a process instance $\hat{p}$ with one repetition of activity c is shown. As described before, conformance checking methods and many process mining techniques assume a one-to-one relation between activities and events on type and on instance level. Consequently, there have to be four event instances with three different event classes in the trace that represents $\hat{p}$. Note that we illustrate event instances from an event log as white bullets with a black double-line border, to distinguish these events from the bullets used in the event diagram for activity instance life cycles. As there is no activity life cycle attached to the activities in Figure 16, we connect the activity instances with the dots of the event diagram for simplicity. The mapping between activity instances and event instances in Figure 16 is represented by a dotted line. Yet, the before mentioned process mining techniques do not take any mapping as input, but require events and activities to share the same labels for matching. The string–based matching between event classes and activities that is assumed by many process mining techniques, is not always possible. There are mainly two reasons for this. First, when retrieving event logs from databases or similar sources without deeper knowledge of the underlying process, it is very unlikely to obtain the same naming of events and corresponding activities. In the worst case, one may be confronted with many rather cryptic event names stemming from the column names of the database tables they have been retrieved from as seen in the order process example in Section 2.3.1. But even if event names are human readable, there might not be a one-to-one match due to different naming. The naming might stem from humans, but can also stem from automated retrieval of events or from an event clustering approach like [75]. In order to have the required naming of events referring to activities, a preprocessing of the event log is necessary. In the example in Figure 16, the trace t therefore has to be transformed to trace $t' = \langle \hat{a_1}, \hat{b_1}, \hat{c_1}, \hat{c_2} \rangle$. For the order process example log in Table 5 the transformation of trace $t_1 = \langle$ O_CHK, O_PRC, I_SM, P_SP, O_ARC $\rangle$ would look as follows: $t'_1 = \langle$ Check order, Process order, Send invoice, Ship products, Archive order $\rangle$. For further reference we define this prerequisite of conformance checking and discovery techniques as Requirement R1.

Requirement R1. *Match activities and events in a one-to-one relationship on type and on instance level.*

Looking at the examples given in Section 2.3, we can see that there are different starting points with regard to the abstraction level of a given event log. For the events from the order process event log shown in Table 5 and the process model in Figure 12, there is exactly one activity for each event class and each event instance represents one activity instance. Yet, the concrete mapping between event classes

and activities is unknown and cannot be established by simple string matching. For the event log displayed by Table 7, we do not find such a one-to-one mapping, as there are more event classes in the event log than activities in the process model. This is another major challenge that can be found in real life settings: events are not on the required abstraction level [118, p. 114]. Table 10 and 11 show the

Table 10: Types of mappings on type level

Relation	Type level
1:1	Exactly one event class represents one activity.
1:n	One activity is represented by different event classes due to lower level of abstraction in the event log.
n:1	Multiple activities are reflected by the same event class because a certain sub–activity is performed in more than one activity.
1:0	An activity is not recorded in the event log.
0:1	Event class is out of scope and cannot be mapped to an activity.

Table 11: Types of mappings on instance level

Relation	Instance level
1:1	One activity instance consists of exactly one event instance.
1:n	An activity instance can be represented by multiple event instances.
n:1	One event instance reflects instances of multiple activities.
1:0	An activity has not taken place.
0:1	Event class is out of scope and event instance cannot be mapped to an activity instance.

different types of relations between events and activities on type and on instance level. An event to activity mapping is always a combination of both a mapping on class and a mapping on instance level. Most often events are more fine–granular than activities found in a business process model, leading to a one-to-many (1:n) relation on both type and instance level. While there are different approaches for abstracting event logs to a higher level, like [58, 59, 75], none of these approaches aims at mapping events to defined activities with a particular abstraction level. Yet, in order to fulfill Requirement R1, it is required to perform an aggregation targeting at the specific abstraction level of the process model that corresponds to the event log at hand. Taking the first trace of the example log shown in Table 7, $t_1 = \langle$ O_CHK_S, O_PR_S, O_PR_E, I_SM_E, P_SP_E, O_ARC_S, O_ARC_E $\rangle$, the transformed and aggregated trace is $t'_1 = \langle$ Check order, Process order, Send invoice, Ship products, Archive order $\rangle$. Note that the two events O_PR_S and O_PR_E have been aggregated to one event "Process order". In the same manner the two events O_ARC_S and O_ARC_E have been aggregated to the event "Archive order". We refer to this requirement as Requirement R2:

Requirement R2. *Match activities and events that are on different abstraction levels.*

Taking different abstraction levels into account, Figure 17 illustrates two different mapping strategies that lead to different discovered process executions. On the bottom line we have the unaltered events that have been extracted from the supporting IT system. In the first step, each event reported by the IT system is mapped to an activity from the designed process model. Applying the mapping to the event log results in a new event log of the same size where all events have been renamed with the names of the corresponding activities they have been mapped to.

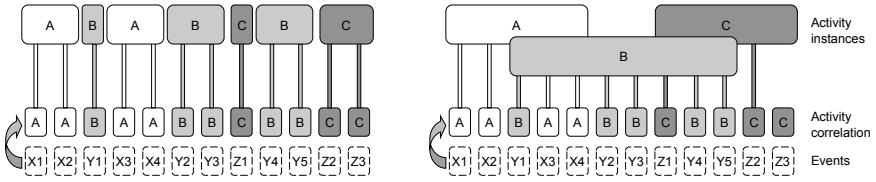


Figure 17: Different abstraction results on the instance level

The fact that multiple event classes may belong to one activity again reflects that event log and process model do not have the same level of abstraction. Thus, the mapped event log contains duplicate entries where different events have been mapped to the same activity name. Looking at the example in Figure 17, we have four instances of activity A in the mapped event log. Existing event log abstraction approaches like [59] assume that one can group all adjacent events mapped to the same activity to one activity instance, as shown in Figure 17a. This, again, leads to several instances of activity a, implying a loop. Yet, if there is concurrency in the process, we could also take the approach to merge all event occurrences of one activity into one activity instance, as illustrated in Figure 17b. Using this strategy raises the problem that we eliminate all loops and rework from the event log. Thereby, concurrency is introduced. This is due to the fact that this strategy does not allow more than one instance of an activity.

Both presented strategies may not reflect reality well and a mapping approach rather needs to support a mixture of loops and concurrency. Therefore, reasonable strategies for dealing with loops and concurrency need to be provided, as stated in Requirement R3.

Requirement R3. *Provide strategies to deal with loops and parallelism.*

Besides a one-to-many relation on type and instance level, Table 10 and 11 also include many-to-one (n:1) relations on both levels. So far, we assumed that event instances of the same event class are mapped to instances of the same activity or activity life cycle transition. Yet, in modern IT systems, activities make use of the same functionalities that are also used by other activities. Consider, for example, a protocol functionality where the process participant logs preliminary or

final results for the currently executed activity. Such a logging might be required for more than one activity. Yet, the IT system might not differentiate between protocol entries for different activities and it remains unclear to which activity an event generated by protocol logging corresponds. Thus, events carrying the same label may belong to different activities. Other examples are given by the incident management process in Figure 14. Here, the event class "CI" is either mapped to the activity "Initial diagnosis", "Investigation and Diagnosis", "Security incident handling", or "Incident closure". In the same manner, the events of class "Group" are assigned to one of the activities "Incident logging", "Functional escalation", "Investigation and Diagnosis", or "Security incident handling". These cases describe a many-to-one relation between activities and events on type level. Besides such cases where different event instances from the same event class refer to different activities, there are cases where a single event instance refers to the execution of different activities, i.e, two different activity instances. For example activities sharing the same data field for documentation. In order to arrive at an event log that can be used by common process mining techniques, such events need to be preprocessed to reflect the correct mapping to their corresponding activities. This requirement is referred to as Requirement R4.

Requirement R4. *Match event instances that reflect shared functionalities to the instances of their corresponding activities or activity life cycle transitions.*

Considering the possible types of relation in Table 10 and 11, there are also one-to-zero (1:0) and zero-to-one (0:1) relationships on each of the two levels. On type level, a one-to-zero relation between events and activities means that an activity is simply not recorded in the event log. This can be due to the fact that the activity is executed without IT system support and thus, no logging is in place. Yet, it can also happen that an IT supported activity is not being logged, e.g., for performance reasons. On instance level, a one-to-zero relation between events and activities means that an activity is simply not executed. We assume that this can only happen if the activity is not recorded at all. Thus, we assume that each event class is found at least once in an event log. If no events are available for an activity, this has to be taken into account by a mapping solution in order to fulfill Requirement R1, for which each activity needs to be mapped to exactly one event on both type and instance level. We refer to this requirement as Requirement R5.

Requirement R5. *Deal with missing events.*

Similar to missing events, are events for which there is no activity (zero-to-one relation). Here, the event classes or event instances are out of scope for the considered process model and need to be filtered

out in order to arrive at a one-to-one mapping on type and instance level. Note that it is possible that there are different relation types on instance and type level. For example, it is possible that an event class represents a shared functionality and is therefore mapped to multiple different activities. Still, not all of its corresponding event instances may be relevant for the process model at hand. Consider the protocol example again. There may be types of protocol entries that are simply not relevant for the process execution of the process model in that an analyst is interested, and thus these event instances cannot be mapped to any of the activities. Nevertheless, other event instances of the same class may well be mapped to activities of the process model. Dealing with such additional events is captured in Requirement R6.

Requirement R6. *Conditionally filter out additional events.*

So far we have looked at the requirements imposed by conformance checking techniques as well as by many process discovery algorithms. Here, a one-to-one relation between events and activities on both type and instance level is required. Turning to performance analysis, a one-to-one relation between event instances and activity instances is not enough as at least two timestamps are necessary to calculate the duration of an activity, i.e., the start and complete timestamps. Figure 18

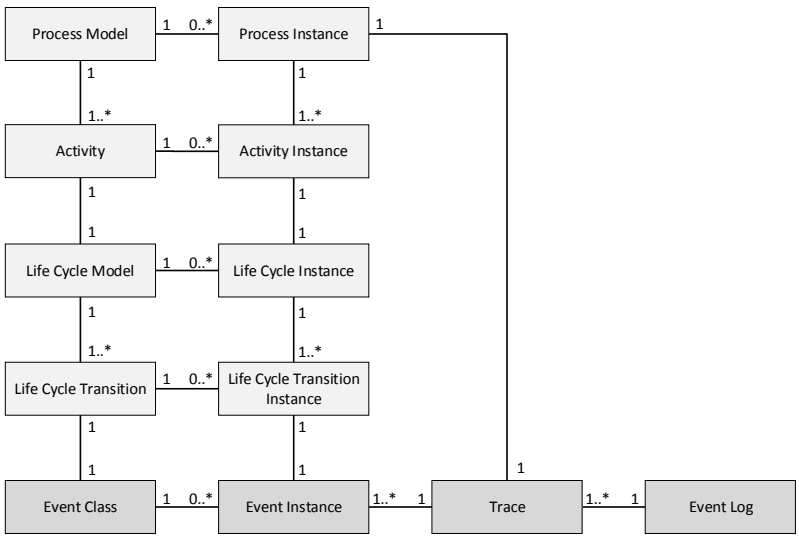


Figure 18: Relations of the entities of event log, process model, and process execution including life cycle transitions with assumed cardinalities

depicts the extended class diagram that also includes the life cycle model of activities and shows the connection between events and life cycle transitions. As explained in Section 2.1.7, an activity has a connected life cycle model with a number of life cycle transitions. Apart from start and complete transitions, also other life cycle transitions might be taken into account. For example, suspending and resuming

of an activity might be considered for the measurement of performance on activity level. Each timestamp needs to be retrieved from a single event instance and related to the corresponding instance of a life cycle transition and thereby, to the corresponding activity instance. Each event instance in the event log represents an instance of a life cycle transition. Thus, each event class needs to be connected to a life cycle transition.

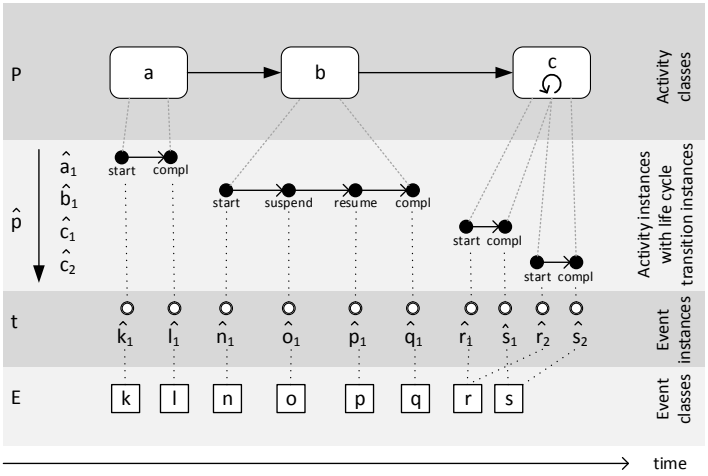


Figure 19: Event classes and instances matched in a one-to-one relation to life cycle transitions and their instantiations

In Figure 19, the previous example from Figure 16 is extended to include the life cycle instances of each activity instance. Each event instance is matched to exactly one instance of an activity life cycle transition. For example, activity instance $\hat{a}_1$ is connected to a start and a complete transition instance in process instance $\hat{p}$. These two life cycle transitions are represented by the events of the event classes k and l. Thus, the two life cycle transition instances connected to $\hat{a}_1$, are represented by the two event instances $\hat{k}_1$ and $\hat{l}_1$ in trace t.

Having only the event log containing trace t and the process model P with connected activity life cycle models, the connection between event instances and activity life cycle transition instances is unknown and needs to be established in order to conduct performance analysis. We denote this requirement as Requirement R7:

Requirement R7. *Match activity life cycle transitions and events in a one-to-one relationship on type and instance level.*

Requirement R7 implicitly includes a one-to-many relation between activity instances and event instances. Such a relation can also be used by process mining techniques to enable zoom-in functionality on activity level as presented for the fuzzy miner in [17]. While Requirement R7 requires a very specific mapping of event instances to activity life cycle transition instances, discovery algorithms with zoom-in functionality do not need the relation of an event instance to

a life cycle transition instance of an activity, but rather to a life cycle transition instance of a sub–activity.

In order to include sub–activities, Figure 20 shows the adopted class diagram that also allows for connections between activities, representing a hierarchical ordering. Note that this connection of activities and sub–activities does not only exist on the model level, but also on the instance level. An example of a matching with sub–activities

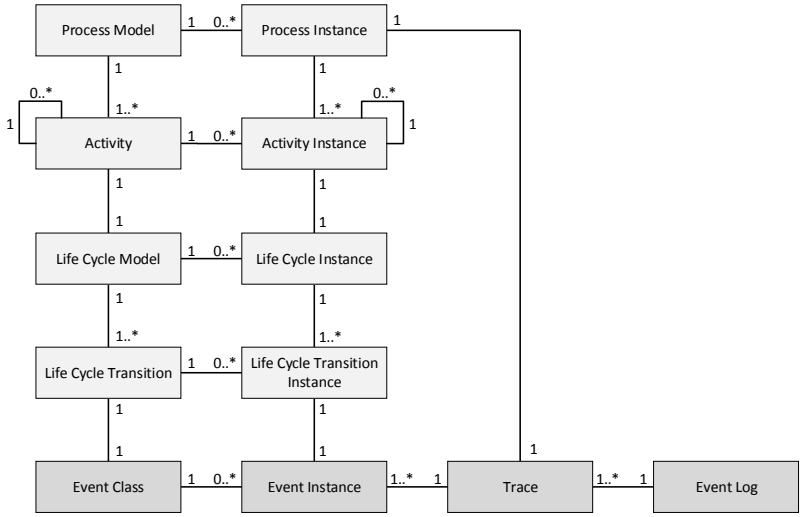


Figure 20: Relations of the entities of the event log, process model, and process execution with assumed cardinalities including activity life cycles and sub–activity relations

is depicted in Figure 22. Here, activity b has two sub–activities ba and bb. Both sub–activities have a life cycle model with a start and a complete transition attached. The life cycle of activity b additionally contains a suspend and a resume transition, which match to the complete transition of ba and the start transition of bb respectively. Depending on the abstraction level we are interested in, the corresponding event instances of event classes n, o, p, q need to be either matched to the life cycle transition instances of activity b or to those of the sub–activities ba and bb.

In a process mining tool, a possible visualization could look as depicted in Figure 21. When selecting an activity, either the connected life cycle model is presented or the model of the sub–activities. Using the timestamps from the connected events, performance information can be plotted onto the model as known, e.g., from the process mining tool Disco. In order to accomplish process discovery with zooming to predefined abstraction levels, the necessary mapping information needs to be included in the event log. Therefore, the events need to be matched to life cycle transition instances of hierarchically related activities. We refer to this requirement as Requirement R8:

Requirement R8. *Match event instances to life cycle transition instances of hierarchically related activities.*

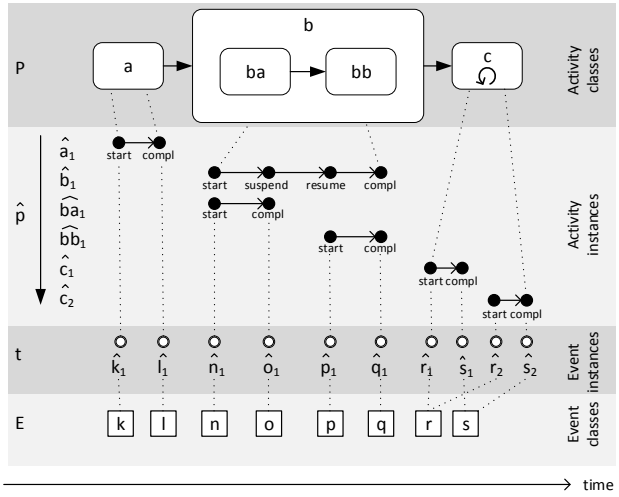


Figure 21: Events matched in a one-to-one relation to life cycle transitions of activities and sub–activities

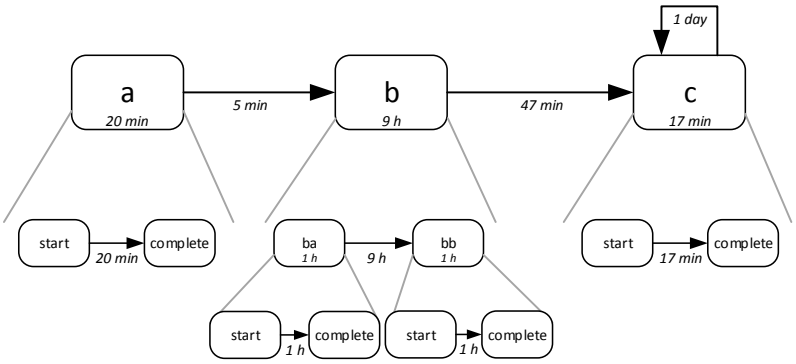


Figure 22: Events matched in a one-to-one relation to life cycle transitions of activities and sub–activities

Considering conformance checking and process discovery, it is obvious that such analyses are only useful if the matching approach is able to deal with nonconforming executions. Non-conformance of process executions to the process model can be either due to willingly or unwillingly incorrect executions by the process participants or due to incorrect logging. As it is not possible for an algorithm to distinguish between incorrect executions and incorrect logging (cf. [118, p. 148]), we assume that the logs we are dealing with, do not contain logging errors. In literature, the term "noise" is used to refer to unwanted or exceptional behavior (cf. [25, 118]). This behavior relates to missing activities, wrong order of activity execution or duplicate activity executions. We furthermore distinguish between behavior that is systematically non-conforming, i.e., patterns of nonconforming behavior that frequently repeat, and nonconforming behavior where no repeating patterns can be found. The incident process introduced in Figure 14 shows an example of systematically nonconforming behavior: in cases where events of the class "Classification"

occur, the events of class "Details" always occur after the events of class "Classification", which behavior is not allowed when considering the mapping to the given process model. In contrast, the order process examples do not show any frequently recurring patterns of nonconforming behavior. Yet, there is a wrong order of execution in one of the traces where the customer has been notified before the products were shipped.

Requirement R9. *Match events to activities where process execution may be nonconforming to the process model.*

This section introduced the major requirements that have to be met to use the different process mining techniques with real life event logs. Table 12 gives a summary of the requirements and introduces their short names, which we will use for easier reference in the text.

Table 12: Overview of the requirements

Short name	Description
R1 1:1 matching to activities	Match activities and events in a one-to-one relationship on type and on instance level.
R2 Different abstraction levels	Match activities and events that are on different abstraction levels.
R3 Loops and parallelism	Provide strategies to deal with loops and parallelism.
R4 Shared functionalities	Match event instances that reflect shared functionalities to the instances of their corresponding activities or activity life cycle transitions.
R5 Missing events	Deal with missing events.
R6 Additional events	Conditionally filter out additional events.
R7 1:1 matching to life cycle transitions	Match activity life cycle transitions and events in a one-to-one relationship on type and instance level.
R8 Hierarchical matching	Match event instances to life cycle transition instances of hierarchically related activities.
R9 Nonconforming execution	Match events to activities where process execution may be nonconforming to the process model.

2.5 RELATED WORK

In this section, we want to elaborate on the work that is related to the research presented in this thesis. In order to structure related work, we have built a literature classification, of which parts have already been published in [8]. On a high level, the mapping between events and activities is similar to the matching problem known from ontologies and data integration [46]. Yet, techniques in this field most often try to use data structures for the matching and do not consider process specific circumstances. Using the internal data structures of events and activities is most often not possible because activities typically do not have further attributes attached, and if they have, these attributes typically are very different from the attributes found in event logs. Therefore, we focus on the research that takes the specifics of business processes into account.

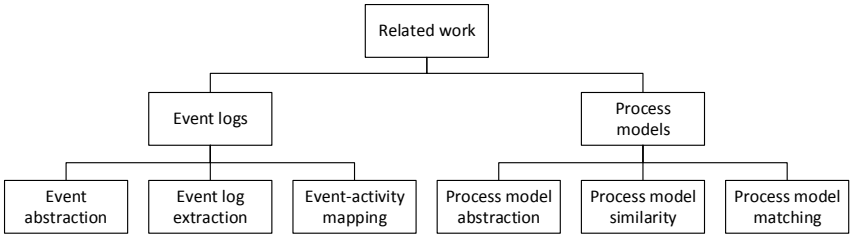


Figure 23: Classification of related work.

As depicted in Figure 23, research related to this thesis can be generally subdivided into approaches working on event logs and approaches working on process models. The work that focuses on event logs can be mainly subdivided into event correlation, event abstraction, event log extraction, and event–activity mapping. The related techniques that work on process models fall into one of three categories: process model abstraction, process model matching and process model similarity. In both main categories — work on event logs and on process models — there are a few hybrid approaches that take both an event log and a process model as input. Yet, they always focus on either log or model when it comes to the objectives and the output of those techniques. An exception is the work by Herzberg et al. [62], which aims at a strong connection between events and process models.

Besides the classification of work into one of the aforementioned categories, we assessed two criteria: The usage of external knowledge and the possible types of relations on class level. The usage of external knowledge shows whether the approach at hand relies on further input beyond process model or event log. A dependency on external knowledge may impose further restrictions because the required external knowledge may not be available. Yet, on the other hand, the usage of external knowledge may also facilitate new and

better means for solving the problem at hand. With this in mind, we examined whether external knowledge is used and if this is the case, we specify what kind of knowledge is used.

With the second criteria, the possible types of relations on class level, we inspect to which extend a technique is able to match artifacts of different abstraction levels. If a technique requires a one-to-one (1:1) relation between matched artifacts, this means they need to be on the same abstraction level. While a one-to-one relation between events and activities is one of the requirements (Requirement R1) for many process mining techniques, it is typically not given in the starting situation. More often, we face one-to-many relations (1:n). When one-to-many relations are supported, we talk about techniques that typically match artifacts from a lower abstraction level to a higher abstraction level. While some process mining techniques require a one-to-many relation, as stated in Requirement R7, being able to match artifacts that are in a one-to-many relation not only facilitates the usage of these approaches, but also enables further processing to achieve a one-to-one relation as required by Requirement R1 when facing different abstraction levels (Requirement R2). Approaches that support many-to-many (n:m) relations are able to deal with settings where there is no clear direction of granularity between two sets of artifacts. In the context of event to activity matching, this is required when different abstraction levels are faced (Requirement R2) and additionally one or more of the event classes represent a shared functionality (Requirement R4).

Table 13 provides the classification of the most relevant related work using the described categories. In the following we will briefly discuss the categorized work with respect to the introduced criteria.

2.5.1 *Approaches Working on Event Logs*

We start with approaches that focus on *event logs* and, in particular, with the *event correlation* approaches. The main objective of event correlation is to assign each event instance to its corresponding process instance. Looking at these techniques only one approach makes use of external knowledge. The work by Pérez-Castillo et al. [86] uses modified sources code as external knowledge to generate candidate correlation attributes. Therefore, this approach requires not only read access to the source code of the supporting IT system, but also needs to modify the source code, which in many situations is not possible or too costly. The relations between the matched objects in event correlation are typically one-to-many relations, since these techniques correlate a set of event instances to a single process instances. Only the work by Steinle et al. [110] considers many-to-many relations by allowing one event instance to belong to multiple process instances. Nevertheless, since the work on event correlation deals with matching

event instances to process instances, it is not suited for the matching of event instances to activities, which are on a much more fine granular level than process instances.

In contrast to this, the approaches that deal with *event abstraction* aim at the matching of event instances to higher level activities. Looking at the possible relations between matched objects on type level, we found that most works in the area of event abstraction support one-to-many relations, which is natural in the context of abstraction.

Günther et al. introduce in [57] an approach that clusters events to activities using a distance function based on time or sequence position. Due to performance issues with this approach, a new means of abstraction on the level of event classes is introduced in [59]. Here, event classes are clustered globally based on co–occurrence of related terms, yielding better performance but lower accuracy. Still, both approaches are only able to relate a particular event class to one specific activity. Thus, they are not able to identify shared functionalities and therefore do not address Requirement R4. The approach introduced by Li et al. [75] can handle many-to-many relations by defining context-dependent patterns and thereby addresses Requirement R4. The approach of Li et al. is based on the work by Bose et al. [15, 17] and also uses co–occurrence of events to find patterns of events that always appear in the same context and are therefore supposed to be semantically related. Yet, context in that work is limited to event classes of surrounding event instances, which is a rather strong restriction as an activity that uses a shared functionality might well have multiple different patterns of surrounding event classes due to choice constructs in the model. Therefore, other means of defining context are necessary. Another shortcoming of the approach by Li et al. is that two events belonging to one activity instance cannot be separated by more than one event of another activity instance in between. This leads to problems when dealing with concurrency as formulated in Requirement R3. Another approach that uses pattern recognition and machine learning techniques for abstraction is introduced in [26]. The authors aim at recognizing activities in streams of sensor data and are able to provide a matching with 60–80 % accuracy. Unfortunately, this approach does not allow to refine mappings to obtain full accuracy and only allows for one-to-many relations.

Folino et al. [52] introduce an approach that aims at building a performance prediction model for traces. They combine event clustering and trace clustering and iteratively improve the clustering with respect to the measured prediction error. They use the values of event attributes for the clustering of event instances, thus, integrating context data into their clustering approach. Yet, the approach does not aim at predefined activities, but looks for the clustering that yields the best performance prediction. A different means of abstracting event logs is

to simply remove insignificant behavior. This requires that a relation between events and activities has already been established. Together with the fuzzy miner, an approach is defined to abstract a mined process model by removing and clustering less frequent behavior [58]. While the clusters are not directly related to activities, they can be interpreted as such. Nevertheless, the approach only allows for one-to-many relations between event classes and clusters. The approach reported in [55] clusters process instances with similar behavior in order to abstract behavior that is different between these clusters. While this is an interesting approach for exploratory analysis, it is not able to abstract events that always occur together.

Furthermore, there are different approaches that apply behavior abstraction in process discovery and trace alignment [16, 47, 92] aiming at better understandable process models by hiding complexity.

In summary, it can be said that all of the presented event abstraction approaches do not aim at predefined activities. Instead, they try to automatically cluster event instances that may belong to the same activity without actually referencing an existing activity.

Looking at the relation of events and existing activities, approaches in the category of event log extraction try to generate events that are related to predefined activities. That is, these approaches do not take an event log as input, but initially produce an event log. Li et al. [76] introduce an approach aiming at intelligently extracting events from IT systems and relating them to automatically identified activities. The authors make intensive use of external knowledge by building process lexica from existing process documents and industry-standard process references. Using the process lexica potential tasks candidates are automatically generated. The process lexica are furthermore used to scan the database of the supporting IT system for relevant attributes and rank them. The list of task candidates and the ranked database attributes are then presented to the analyst to identify the process mining related information. The matching between the activities and database attributes is done manually by the process analyst. As there is no further mechanism to deal with attributes that relate to multiple activities, the approach only supports one-to-many relations. Another approach in the category of event log extraction is the technique proposed by Pérez-Castillo et al. in [85]. In this work, the authors manipulate the source code of the supporting IT system in order to generate events that belong to predefined business activities. Again, the authors do not provide any mechanism to deal with shared functionalities and therefore only support one-to-many relations. As this technique requires deep control and manipulation of the supporting IT system, it is only feasible in very few settings.

The last category in the group of approaches focusing on event logs is the *event to activity mapping* category. Here, existing events are mapped to predefined activities from a process model. So far, we could only identify one work in this specific category. Herzberg et al. [62] present a framework to correlate events to predefined process event monitoring points (PEMP) in a process model. These PEMPs are attached to the life cycle of activities. The focus of their work is on the conceptual framework of event correlation to business processes and touches only the correlation of events to process instances in a concrete manner, using ESPER or SQL queries. They do not provide concrete mechanism for the relation of events to PEMPs. While the work in [62] conceptually considers many-to-many relations between activities and events, it assumes that event instances of event classes that are related to multiple activities, always reflect the execution of all related activities. Thus, it is not possible to have context-dependent assignment of events from the same event class to multiple activities.

2.5.2 *Approaches Working on Process Models*

Having elaborated on the approaches that focus on event logs, we turn to the second category, i.e., approaches that concentrate on *process models*. Process model techniques are more advanced when it comes to the usage of external knowledge. Especially in recent years, more sophisticated techniques that use linguistic information have evolved [71, 66]. Other model-based techniques — as, e.g., in [106] — leverage different semantic information as, e.g., roles, resources or data objects, which can be seen as external knowledge in this area. In this thesis, we built on these works by also leveraging different types of external knowledge. While we use linguistic information in the matching of events and activities, we furthermore make use of process descriptions to extend the available information about activities in the process model.

Pointing the focus to the possible relations between matched objects, it can be seen that there are differences in the particular subcategories of model-based approaches. For all subcategories, the matched objects are activities only. That is, activities are matched to other activities. In the works on process model abstraction we typically find one-to-many relations between the matched activities. Model similarity tries to make one-to-one relations between the sets of activities of two different process models. In the area of process model matching there are several works that establish many-to-many relations [20, 135, 140]. The work on ICoP defines a generic framework for process model matching [135]. This framework is extended with semantic concepts and probabilistic optimization in [71], adapting general concepts from ontology matching [46]. The implications of different abstraction levels for finding correspondences are covered in [66, 140, 141]. In [67],

Klinkmüller et al. introduce an iterative mixed–initiative approach for the matching of activities from different process models. To this end, they automatically derive potential matches using different indicators such as position, neighborhood, label specificity, label and execution semantics. The user is provided with the discovered correspondences and has to decide on correctness and add missing matches. Based on these decisions the matching algorithm is updated for the next matchings. However, all these works focus on finding matches between two process models, not between events and activities.

2.5.3 *Summary of Related Work*

To conclude the research on related work, it can be said that there is an already quite large body of work that deals with similar topics in the field of business process management. Nevertheless, there are only very few works that directly focus on the matching of events to predefined activities. Only the work by Herzberg et al. specifically looks at the binding of events to activities in a process model. Yet, as most of the discusses approaches, it only provides limited possibilities to handle Requirement R4 (Shared functionalities) and gives rather conceptual guidance than automated support for the matching of events and activities. To this respect, this thesis advances the state of the art providing the formal and technical background for the matching as well as automated support to assist the process analyst.

Table 13: Overview of related work.

Authors (Year)	Title	Approach	Category	Model input	Event log input	External knowledge	Class level relation
Beheshti et al. (2011) [12]	A Query Language for Analyzing Business Process Execution	Related events are grouped into folders (process instances) and paths (models) by correlation conditions expressed in an extension of SPARQL	event correlation	no	yes	no	1:n
Bose et al. (2013) [18]	Enhancing Declare Maps Based on Event Correlations	Pruning relations between events not sharing the same data objects and disambiguation of event relations using conditions on their attributes	event correlation	yes	yes	no	1:n
Motahari-Nezhad et al. (2010) [82]	Event correlation for process discovery from web service interaction logs	Correlating event instances to process instances using attribute values	event correlation	no	yes	no	1:n
Pérez-Castillo et al. (2012) [86]	Assessing event correlation in non-process-aware information systems	Discovering correlation sets over attributes from events generated by inserting source code statements	event correlation	no	yes	source code	1:n
Rozsnyai (2011) [103]	Discovering Event Correlation Rules for Semi-Structured Business Processes	Discovery of event correlation rules based on statistics of event attributes	event correlation	no	yes	no	1:n
Steinle et al. (2006) [110]	Mapping Moving Landscapes by Mining Mountains of Logs : Novel Techniques for Dependency Model Generation	Correlating event classes (here systems) by time proximity and user sessions	event correlation	no	yes	no	n:m
Bose et al. (2009) [15]	Abstractions in process mining: A taxonomy of patterns	Clustering of correlated event classes	event abstraction	no	yes	no	n:m
Bose et al. (2011) [17]	Discovering Hierarchical Process Models Using ProM	Hierarchical clustering of correlated event classes	event abstraction	no	yes	no	n:m
Cook et al. (2013) [26]	Activity Discovery and Activity Recognition: A New Partnership	Mapping sensor data to activities using machine learning techniques	event abstraction	no	yes	no	n:m

Continued

Authors (Year)	Title	Approach	Category	Model input	Event log input	External knowledge	Class level relation
Folino et al. (2014) [52]	Mining Predictive Process Models out of Low-level Multidimensional Logs	Clustering event instances to activities using event attributes; targeting optimal performance prediction	event abstraction	no	yes	no	1:n
Günther et al. (2006) [57]	Mining Activity Clusters From Low-level Event Logs	Clustering correlated event instances based on proximity on trace level	event abstraction	no	yes	no	1:n
Günther et al. (2007) [58]	Fuzzy mining: adaptive process simplification based on multi-perspective metrics	Clustering correlated event classes; omitting infrequent event classes	event abstraction	no	yes	no	1:n
Günther et al. (2009) [59]	Activity mining by global trace segmentation	Hierarchical clustering of correlated event classes	event abstraction	no	yes	no	1:n
Li et al. (2011) [75]	Mining context-dependent and interactive business process maps using execution patterns	Hierarchical clustering of correlated event classes; omitting of insignificant event classes	event abstraction	no	yes	no	n:m
Li et al. (2015) [76]	An intelligent approach to data extraction and task identification for process mining	Automated task identification and derivation of relevant database attributes for process mining using generated process lexica	event log extraction	no	no	process documents, reference processes	1:n
Pérez-Castillo et al. (2011) [85]	Generating event logs from non-process-aware systems enabling business process mining	Generating events that reflect business activities by inserting source code statements	event log extraction	no	no	source code	1:n
Herzberg et al. (2013) [62]	Improving the Understanding of BAM Technology for Real-Time Decision Support	Platform for enriching of raw events and correlation to process execution using manually defined queries	event to activity mapping	yes	yes	context data	1:n
Fahland et al. (2013) [47]	Simplifying Discovered Process Models in a Controlled Manner	Generalize behavior of a model using log-induced unfoldings and model structure simplifications	model abstraction	yes	yes	no	n/a
Greco et al. (2008) [55]	Mining taxonomies of process models	Abstraction of activities from hierarchical clustered process models mined using trace clustering	model abstraction	yes	yes	no	1:n

Continued

Authors (Year)	Title	Approach	Category	Model input	Event log input	External knowledge	Class level relation
Leopold et al. (2014) [73]	Simplifying process model abstraction: Techniques for generating model names	Construct name proposal from the text labels of a process model	model abstraction	yes	no	synonym relations and Lin metric	n/a
Polyvyanyy et al. (2008) [92]	Process Model Abstraction: A Slider Approach.	Aggregation and elimination of insignificant model elements	model abstraction	yes	no	probabilities	1:n
Polyvyanyy et al. (2009) [93]	On Application of Structural Decomposition for Process Model Abstraction	Hierarchical structural decomposition of process models	model abstraction	yes	no	no	1:n
Smirnov et al. (2012) [106]	From fine-grained to abstract process models : A semantic approach	Activity aggregation by clustering with a distance measure over the properties of activities	model abstraction	yes	no	roles, data	1:n
Smirnov et al. (2013) [107]	Business Process Model Abstraction Based on Synthesis from Well-Structured Behavioral Profiles.	Leveraging behavioral profiles to derive control–flow structure for abstracted models	model abstraction	yes	no	no	1:n
Branco et al. (2010) [20]	Matching business process workflows across abstraction levels	Matching of model fragments based on attributes and control flow structure using process structure trees.	model matching	yes	no	no	n:m
Klinkemüller et al. (2014) [67]	Listen to me: Improving Process Model Matching through User Feedback	Matching of process model activities in an iterative mixed–initiative approach based on different indicators and user feedback.	model matching	yes	no	no	n:m
Klinkemüller et al. (2013) [66]	Increasing Recall of Process Model Matching by Improved Activity Label Matching	Matching based on bag-of-words and label pruning	model matching	yes	no	semantic relations (WordNet, Lin metric)	1:1
Leopold et al. (2012) [71]	Probabilistic Optimization of Semantic Process Model Matching	Matching based on annotation, semantic relations and behavioural constraints	model matching	yes	no	semantic relations (WordNet, Lin metric)	1:1
Weidlich et al. (2010) [135]	The ICoP Framework : Identification of Correspondences between Process Models	Matching based on syntactic label similarity	model matching	yes	no	no	n:m

Continued

Authors (Year)	Title	Approach	Category	Model input	Event log input	External knowledge	Class level relation
Dijkman et al. (2011) [39]	Similarity of business process models: Metrics and evaluation	Matching of models using semantic label matching, structural and behavioral matching	model similarity	yes	no	synonym relations	1:1
Dongen et al. (2008) [131]	Measuring Similarity between Business Process Models	Matching of models based on labels and input/output context	model similarity	yes	no	synonym relations	1:1
Kunze et al. (2011) [69]	Behavioral similarity - a proper metric	Similarity measure of two models based on the Jaccard coefficient using behavioral profiles	model similarity	yes	no	no	1:1
Polyvyanyy et al. (2012) [94]	Isotactics as a Foundation for Alignment and Abstraction of Behavioral Models	Equivalence of aligned models with complex correspondences using concurrency semantics	model similarity	yes	no	no	n:m
Weidlich et al. (2012) [140]	Behaviour Equivalence and Compatibility of Business Process Models with Complex Correspondences	Determining behavioral equivalence between sets of activities	model similarity	yes	no	no	n:m

Part II

APPROACHES TO MATCH EVENTS AND ACTIVITIES

3

BASE APPROACH

This chapter introduces our base approach to the mapping problem. The presented work has partly been published in [6, 7, 8]. The base approach provides a framework to fulfill all presented requirements. We will briefly recap the made assumptions in Section 3.1. In Section 3.2, the matching problem is formalized. The focus lies on the complete description and formalization of the mapping problem with all its facets coming from the different presented requirements. Based on this formalization, the concrete steps and concepts of the base approach are introduced in Section 3.3. While the base approach does not aim at a high degree of automation for the type–level matching, it does deliver a complete initial approach for the mapping of events to activities on type level and provides automated mapping on instance level once the type–level mapping is provided. The approaches that are presented in later chapters are grounded on the base approach and add possible automation of the type–level matching for certain specific scenarios.

3.1 REQUIREMENTS AND ASSUMPTIONS

As input for the base approach we assert to have a set of activities, A, as well as an event log L. The set of activities may come from an existing process model but could potentially also be manually provided by an analyst without a model. For simplicity, we assume that the activities are contained in a process model. Note that this does not impose any restrictions in terms of available data, because the base approach will not make use of the flow relation of activities. Thus, a process model could easily be created using only the user defined activities without any flow relation. Activities may be related by the $subAct$ function to implement a hierarchical composition. The event log L has to follow the structures described in Section 2.1.8. Note that this also includes additional event attributes. The base approach is able to make use of additional non-standard attributes, but it does not explicitly require their availability. Furthermore, we assume that event log L contains executions of the set of activities A. We do not make any further assumptions about the properties of the event log.

3.2 THE MATCHING PROBLEM FORMALIZED

The goal of the base approach is to provide a generic approach that fulfills all requirements listed in Section 2.4. This section formally defines the matching problem in the light of the defined requirements.

As a starting point, Definition 7 defines the general relations of events and activities on type and instance level.

Definition 7 (Activity event relations). *Let* $P = (N, \omega, \mu)$ *be a process model with* $A \in N$ *as the non-empty set of activities. Let* $L \in \mathbb{B}(\hat{E}^*)$ *be an event log that records the execution of the process described by P, with* E *being the set of contained event classes in* L*. Then* $AE \subseteq A \times E$ *is the relation that maps activities to their corresponding event classes.* $\hat{A}\hat{E} \subseteq \hat{A} \times \hat{E}$ *is the relation which assigns every event instance from the event log* L *to an activity instance of process instance of* P*.* ◇

Requirement R1 demands that the relation of event instances to activity instances as well as the relation of event classes to activity types follows a one-to-one relationship. Regarding Requirement R2 and R4, it is obvious that the relations AE and $\hat{A}\hat{E}$ do not describe one-to-one relationships of activities and events when events are on different abstraction levels and events potentially represent shared functionalities. Events are most often on a more fine granular abstraction level than activities, leading to a one-to-many relation between activities and events. That means, one activity in the process model is represented by events from multiple event classes in the event log. Therefore, also a single activity instance may relate to multiple event instances. Shared functionalities lead to event classes that are related to multiple activities. As shared functionalities can occur on type and instance level, they lead to many-to-one relations between activities and events on both levels. Thus, shared functionalities in combination with different abstraction levels lead to a many-to-many relation between activities and event classes.

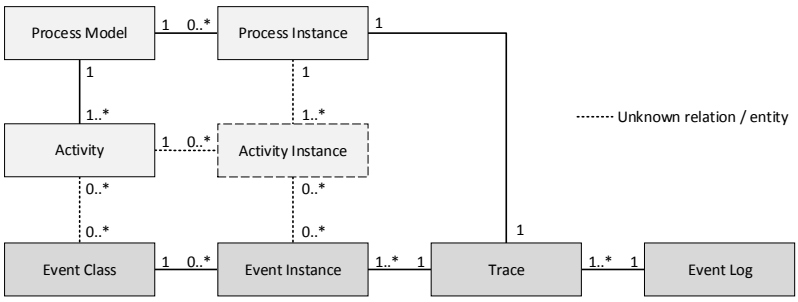
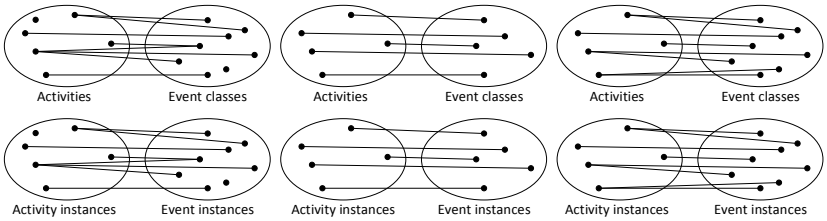


Figure 24: Relations of the entities of the process model and process execution recorded in real life logs that are on a different abstraction level and contain shared functionalities.

Figure 24 depicts the relations between process models and event logs with the cardinalities faced in real life logs. In order to use any process mining technique, the event instances found in the traces of the event log need to be connected to activity instances. Yet, activity instances are not known without the event log. While this connection can be partly established using the link from event instances to event classes and the connection between model activities and event classes,

typically, the latter connection is also unknown. As the relation on type level is a lot smaller than the relation on instance level in terms of the number of entities that have to be matched, it can often be created manually with reasonable effort. Chapter 4, 5 and 6 all deal with semiautomatic approaches to retrieve the relation AE.



(a) Real life log (b) Requirement R1 (c) Requirement R7, R8

Figure 25: Relations of the sets of activities and events on type and instance level.

Figure 25 depicts a comparison of the different types of relations between the set of activities and the set of events, i.e., the different cardinalities for relations AE and $\hat{A}\hat{E}$. Here, one can see the difference between the relations found in real life logs and those required by process mining techniques. As shown in Figure 25a, the relations AE and $\hat{A}\hat{E}$ potentially describe many-to-many relationships for real life logs. Hence, the event log L has to be transformed to an event log L' such that both relations AE and $\hat{A}\hat{E}$ are bijective functions, in order to fulfill Requirement R1. The transformed event log allows for a bijective function $AE_{1:1} : E \to A$, which assigns each event class to exactly one activity, as depicted in Figure 25b. There is no activity that is not matched to an event class and no event class that is not matched to an activity. Hence, all events in an event log have to be either mapped onto their corresponding activity or removed from the log. Knowing to which activity an event instance belongs does not yet reveal to which instance of that activity the event instance belongs, as there may be multiple instances of that activity. Requirement R1 requires a bijective function $\hat{A}\hat{E}_{1:1} : \hat{E} \to \hat{A}$, which assigns each event instance in the transformed event log to exactly one activity instance. Thus, techniques have to be defined to derive the relation $\hat{A}\hat{E}_{1:1}$, which essentially informs about the existing activity instances. Having the function $\hat{A}\hat{E}_{1:1}$, the given event log can be transformed to be used by conformance checking or discovery methods. Therefore, a transformation function $transform_{EA} : L \times \hat{A}\hat{E}_{1:1} \to L$ is needed.

Requirement R7 (1:1 matching to life cycle transitions) and Requirement R8 (Hierarchical matching) demand a one-to-many relation between events and activities as shown in Figure 25c. In contrast to Requirement R1, which requests a direct mapping from events to activities, Requirement R7 and R8 request an assignment of events

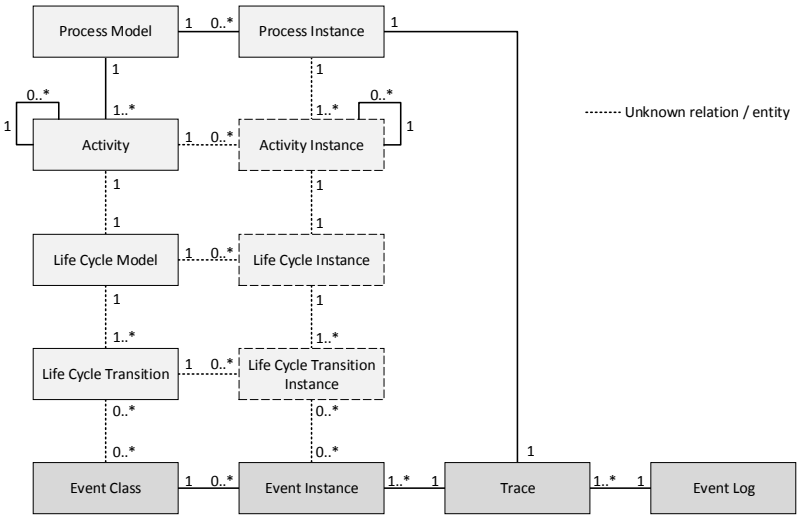


Figure 26: Relations of the entities of the event log, process model, and process execution including life cycle transitions.

to activity life cycle transitions as needed for performance analysis. Again, a one-to-one relation is needed on both type and instance level. We denote the corresponding relations as $LTE \subseteq LT \times E$ and $\hat{LTE} \subseteq \hat{LT} \times \hat{E}$. Similar to the direct matching of events to activities for Requirement R1, we need to end up with two bijective functions that describe the one-to-one matching. On type level, the bijective function $LTE_{1:1} : E \to LT$ assigns each event class to exactly one life cycle transition. On instance level, the bijective function $\hat{LTE}_{1:1} : \hat{E} \to \hat{LT}$ assigns each event instance to exactly one instance of a life cycle transition.

Figure 26 depicts the relations between process models and event logs including the life cycle entities with the cardinalities found in real life scenarios. In the same manner as explained before, we are facing a many-to-many relation between life cycle transitions and event classes due to shared functionalities and different abstraction levels. Due to the before mentioned fact that activity instances are unknown without the connection of process model and log, also the life cycle instances and the instances of the life cycle transitions are unknown. Similarly to the direct matching of activities and events, the relations can be concluded by establishing the links between events and activity life cycle transitions on type level, i.e., by deriving the relation LTE, which also has to be transformed from a many-to-many relation to the bijective function $LTE_{1:1}$ (see Figure 25c). Having the function $LTE_{1:1}$, techniques have to be established to derive $\hat{LTE}_{1:1}$, which informs about the instances of life cycle transitions and thereby about life cycle instances and activity instances. Using the function $\hat{LTE}_{1:1}$, a transformation function $transform_{ELT} : L \times \hat{LTE}_{1:1} \to L$ can be defined to preprocess the event log for performance analysis techniques.

The class diagram depicted in Figure 26 also contains the hierarchical relations between activities. Requirement R8 demands to match events to those hierarchically related activities. This matching can be done using multiple different mapping functions, one for each abstraction level of the activities. Yet, to actually reflect the hierarchical relation in the transformed event log, another relation is required that relates event instances mapped to an activity A to the traces of event instances that are mapped to sub–activities of A. To this end, a new event log L_a is created for each activity $a \in A$. The function $subLog : A \to L$ captures the relation from activities to their corresponding event log. The event log L_a related by $subLog(a)$ contains a trace $t_{\hat{a}}$ for each activity instance $\hat{a}$ of activity a. Such a trace entails all event instances that are mapped to activity instance $\hat{a}$, i.e., for each $\hat{e} \in t_{\hat{a}}$ it holds that $\hat{AE}(\hat{e}) = \hat{a}$ or respectively $\hat{LTE}(\hat{e}) = \hat{lt}$ with $\hat{lt}$ being a life cycle transition instance of $\hat{a}$.

3.3 PHASES OF THE BASE APPROACH

The previous section formally defined the matching problem. This section introduces our base approach to tackle the matching problem. The approach consists of four distinct phases, which together address the defined requirements:

1. Matching of activities and events on type level.
2. Definition of context–sensitive mappings.
3. Transformation of the event log.
4. Clustering of event instance to activity instances.

Figure 27 shows the four phases of the base approach with their inputs and outputs. In the first phase, the relation between activities and events is established on type level, i.e., the relation AE is defined. The second phase deals with the handling of shared functionalities requested by Requirement R4. Context–sensitive mappings are defined in order to define under which contextual circumstance an event is mapped to a particular activity in case its event class is mapped to multiple activities in AE. Furthermore, specific life cycle mappings can be defined if necessary, leading to a partly defined LTE relation. The third phase transforms the given event log using the previously defined AE relation as well as the LTE relation where applicable. The transformation leads to the relation of event instances to activity life cycle transitions ($\hat{LTE}$). During the last phase, the transformed event log is again processed to cluster event instances to their corresponding activity instances. The clustering phase uses user–defined definitions for identifying where an activity instance starts and where it ends. It thereby automatically derives start and complete transitions

of the activity instance life cycle. Thus, the relation L̂T̂Ê is derived in this phase, thereby deducing ÂÊ, the relations of event instances to activity instance. Moreover, the last phase also completes the type–level relation of events and activity life cycle transitions (LTE). This phase also controls whether the ÂÊ relation is a one-to-one relationship or one-to-many relationship, depending on the required result. The following sections will detail each of the four phases.

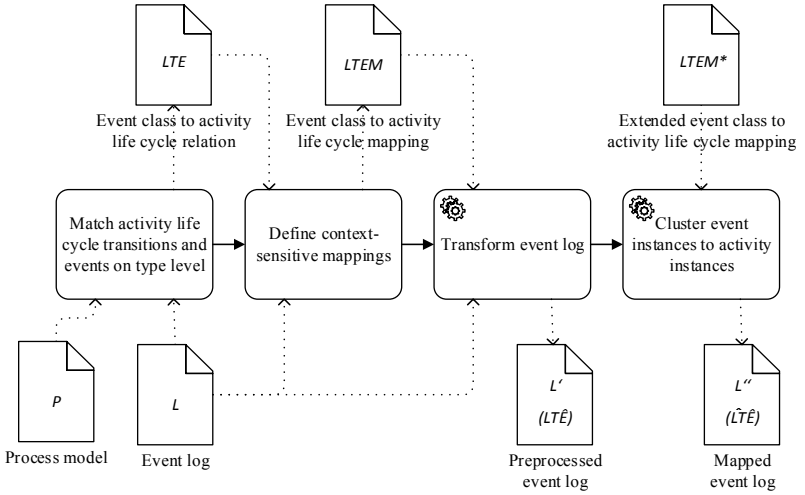


Figure 27: Overview of the base approach for mapping events to defined activities including inputs and outputs for each phase.

3.3.1 *Matching of Activities and Events on Type Level*

The goal of this phase is to establish the type–level relations between events and activities (AE), or between events and life cycle transitions (LTE). For the base approach we assume that the relations AE and LTE can be provided manually by an analyst. As explained before, different process mining techniques have different requirements. Some request for a mapping of events to life cycle transitions and some do not. Yet, having the mapping of events to life cycle transitions of activities entails the mapping of events to activities. We will introduce an approach that is able to automatically assign the two common life cycle transitions, start and complete, automatically. The user is requested to assign all other life cycle transitions. But this assignment is only requested, if it is needed to fulfill the requirements of the process mining technique for which an event log is preprocessed. The approach will automatically assign start and complete transitions and tag all other events with a special life cycle transition, named *execute*, if no specific mapping is defined by the user. Hence, if the analyst does not need any special life cycle transitions other than start and complete, they do not have to specify any relation to life cycle transitions, but only to the activities. Therefore, we will from

now on work with the relation LTE and disregard the relation AE, which can be derived from LTE. In order to reflect assignments that are derived automatically in a later phase, we introduce the place holder ϕ for the set of automatically derivable life cycle transitions $\{start, complete\}$. Thus, the set of life cycle transitions for this phase of the matching is $LT = LT \cup \phi \setminus \{start, complete\}$.

While the main goal is to map events to activities, there may be events in an event log which cannot be mapped to an activity, because they are out of scope. Thus, we need to abstract from them. While some of the existing event log abstraction approaches, like [58], implement this type of abstraction by automatically hiding events from infrequent event classes, we observed that it is often better to let a domain expert control what to omit and what to keep. Sometimes, events that occur very often are not interesting from a business point of view and infrequent events can be highly important once they occur. In some cases one might also not want to omit all events belonging to a specific event class, but only those event instances fulfilling certain conditions. We therefore introduce an activity place holder for LTE, called *REMOVE_EVENT*. The next section will show how context–sensitive mappings can be defined that can also be used to conditionally remove events. In the same fashion, it can be helpful to abstract from complete traces that contain certain behavior identified by event instances in a certain context. The activity place holder for removing complete traces is called *REMOVE_TRACE*.

As a last step of the type–level matching, activities for which the execution is not captured in the event log of the supporting IT system need to be identified. These activities can either be removed from the process model or turned into silent transitions in the Petri net representation. By that, they will not be considered for the mapping and in further process mining analyses, such as conformance checking or performance analysis.

Table 14 provides an example for type–level relation of events to activity life cycle transitions, which is the result of this phase. The relation is taken from our incident process example. Every event class except for the "Message" event class is assigned to the ϕ life cycle transition of an activity. By that, event classes are not directly assigned to a life cycle transition but only to a place holder that is replaced later during the mapping. The event class "Message" is mapped to the *REMOVE_EVENT* placeholder as it cannot be related to any activity of the incident process. One can also see that event classes are mapped to multiple activities, as, e.g., "CI" and "Group". These represent shared functionalities. On the other hand, there are also activities, such as "Incident Logging", that occur in multiple tuples. This shows that these activities are on a higher abstraction level than their corresponding events.

Table 14: Example of the type–level relations between events and activity life cycle transitions (LTE) for the incident process.

Event class	Activity	Life cycle transition
CI	Incident closure	ϕ
CI	Initial diagnosis	ϕ
CI	Investigation and diagnosis	ϕ
CI	Security incident handling	ϕ
Classification	Incident classification	ϕ
Details	Incident logging	ϕ
Group	Functional escalation	ϕ
Group	Incident logging	ϕ
Group	Investigation & diagnosis	ϕ
Group	Security incident handling	ϕ
Message	*REMOVE_EVENT*	
Solution	Resolution and recovery	ϕ
Status	Incident closure	ϕ

3.3.2 *Definition of Context–sensitive Mappings*

As stated in Requirement R4, there are often events representing shared functionalities, which are used by multiple activities. Hence, the event–activity relations on type level cannot be directly used for the mapping of event instances, as we have to disambiguate the event instances for which there are multiple relations of their corresponding event class in LTE. This section describes the necessary steps to get from the relations to a concrete event–activity mapping that can be used to transform the event log. The challenge in this context is to identify the conditions that help to decide when one event class matches one of alternative activities. To this end, we consider the context of an event instance, either as defined over the event or trace attributes or over the surrounding event instances.

First, we take *attributes* into account. Table 15 shows a more detailed event log for the incident process. The log includes a timestamp, the executing role, and the new value that has been assigned for each event instance. Furthermore, the correct activity has been annotated to illustrate the goal for the current phase. From the last phase it is known that the selection of a configuration item (CI) signals the execution of either one of the activities "Initial diagnosis", "Investigation and diagnosis", "Security incident handling", or "Incident closure". Looking at Figure 14, it can be seen that the differentiation can be made partly using the role attribute. The selection of a configuration item belongs to the activity "Investigation and diagnosis" when executed by a second level agent while event instances of the same event class refer to the activity "Security incident handling" when executed by a member of the security role. When an event instance of class "CI" occurs with the change executed by a first level supporter, it belongs

Table 15: Example log of the incident process with mapping to activities.

Trace	Event class	Value	Time stamp	Role	Activity
t_1	Group	1st-SAP	04.02.14 12:31	1st level	Incident logging
t_1	Classification	SAP	04.02.14 12:35	1st level	Incident classification
t_1	Details	SAP is offline	04.02.14 12:36	1st level	Incident logging
t_1	Details	SAP R3 is offline	04.02.14 12:45	1st level	Incident logging
t_1	Solution	Cleared cache	04.02.14 13:31	1st level	Resolution and recovery
t_1	Status	Closed	04.02.14 13:33	1st level	Incident closure
t_2	Group	1st-Intra	04.02.14 12:51	1st level	Incident logging
t_2	Classification	Password	04.02.14 12:54	1st level	Incident classification
t_2	CI	US1234	04.02.14 12:55	1st level	Initial diagnosis
t_2	Details	Password forgotten	04.02.14 12:59	1st level	Incident logging
t_2	Solution	Password reset	04.02.14 13:09	1st level	Resolution and recovery
t_2	Status	Closed	04.02.14 13:10	1st level	Incident closure
t_3	Group	1st-Mail	04.02.14 14:29	1st level	Incident logging
t_3	Classification	Mail	04.02.14 14:35	1st level	Incident classification
t_3	CI	Outlook Client	04.02.14 14:37	1st level	Initial diagnosis
t_3	CI	Outlook Server 1	04.02.14 14:39	1st level	Initial diagnosis
t_3	Details	Cannot send mails	04.02.14 14:44	1st level	Incident logging
t_3	Group	2nd-Mail	04.02.14 14:45	1st level	Functional escalation
t_3	CI	Outlook Server 2	04.02.14 14:46	2nd level	Investigation and diagnosis
t_3	Group	1st-Mail	04.02.14 15:21	2nd level	Investigation and diagnosis
t_3	Solution	Change client settings	04.02.14 15:36	1st level	Resolution and recovery
t_3	CI	Outlook Client	04.02.14 15:46	1st level	Incident closure
t_3	Status	Closed	04.02.14 15:56	1st level	Incident closure
t_4	Group	1st-generic	04.02.14 12:53	1st level	Incident logging
t_4	CI	Notebook 325	04.02.14 12:54	1st level	Initial diagnosis
t_4	Details	Virus found	04.02.14 12:55	1st level	Incident logging
t_4	Group	Security	04.02.14 12:56	1st level	Functional escalation
t_4	CI	Station 133	04.02.14 12:57	Security	Security incident handling
t_4	CI	Win-LP231	04.02.14 12:57	Security	Security incident handling
t_4	Group	1st-generic	04.02.14 13:35	Security	Security incident handling
t_4	Solution	Virus removal	04.02.14 13:42	1st level	Resolution and recovery
t_4	CI	Win-LP232	04.02.14 13:51	1st level	Incident closure
t_4	Status	Closed	04.02.14 13:59	1st level	Incident closure
t_5	Group	1st-monitor	05.02.14 08:17	1st level	Incident logging
t_5	Classification	Backup	05.02.14 08:23	1st level	Incident classification
t_5	CI	Backup Server 1	05.02.14 08:25	1st level	Initial diagnosis
t_5	CI	HD 142	05.02.14 08:27	1st level	Initial diagnosis
t_5	Details	Disk space shortage	05.02.14 08:32	1st level	Incident logging
t_5	Solution	Changed broken HD	05.02.14 08:33	1st level	Resolution and recovery
t_5	CI	HD 157	05.02.14 08:43	1st level	Incident closure
t_5	Status	Closed	05.02.14 08:53	1st level	Incident closure
t_6	Details	High network traffic	06.02.14 06:34	1st level	Incident logging
t_6	Group	1st-monitor	06.02.14 06:35	1st level	Incident logging
t_6	Solution	Short peak. No action	06.02.14 06:55	1st level	Resolution and recovery
t_6	Status	Closed	06.02.14 06:59	1st level	Incident closure

either to the activity "Initial diagnosis", or to the activity "Incident closure". In this case, the differentiation cannot be made on the role or any other attribute attached to any event instance. While the selection of a CI normally happens during the activities "Initial diagnosis", "Investigation and diagnosis", or "Security incident handling", it can also be performed as a quality improvement step during the closure of the incident ticket by a member of the first level role. This is always the case if the solution has been documented before the event "CI" occurs, as shown in Figure 14 and also visible in Table 15. Also, an event instance might be interpreted differently if it occurs for the first time or if it has been preceded by earlier executions of event instances from the same event class. In the example in Figure 14 and Table 15, the working group is always set in the beginning where it refers to the logging of the incident while every other change of the working group refers to a functional escalation or the handover made in the end of the activities "Investigation and diagnosis" and "Security incident handling" depending on the executing role.

Concluding our examples, the relation of an event instance to an activity might depend on attribute data as well as on the *context* in terms of preceding or succeeding event instances. In order to use such domain knowledge it has to be encoded in a formal way. We therefore introduce attribute conditions and event context conditions in Definitions 8 and 9.

Definition 8 (Attribute condition). *Let $AT = ATE \cup ATT$ be the set that unites all event and trace attributes. Let O be the set of comparison operators and let V be the set of values that an attribute $attr \in AT$ should be tested against. Then, an attribute condition is a tuple $ac \in AT \times O \times V$. AC is the set of all attribute conditions. An attribute condition ac is evaluated for an event instance by the function $EV_{ac}(ac, \hat{e}) = o(\#_{attr}(\hat{e}), v)$, where $o \in O$ is a boolean function that compares two given input values, and $v \in V$ is the value given by the attribute condition.* ◇

Definition 9 (Event context condition). *Let T bet the set of all traces. The event context EC for an event instance $\hat{e}$ is defined as $EC(\hat{e}) = (t_{before}, t_{after})$ such that $\exists t \in T : t = t_{before} \| \hat{e} \| t_{after}$ where t_{before} and t_{after} are sub-traces of trace t. The sub-traces can be accessed by the function $EC(\hat{e}, r) \rightarrow T$, where $r \in \{before, after\}$ refers to the part of an event context EC. $EC(\hat{e}, before)$ returns the sub-trace t_{before} and $EC(\hat{e}, after)$ returns the sub-trace t_{after}.*

A condition over a trace is defined by a function $f : T \rightarrow \{true, false\}$, which evaluates a linear temporal logic (LTL) formula [91]. The set of all LTL formula functions is referred to as F.

An event context condition is a tuple $ecc \in F \times \{before, after\}$. ECC is the set of all event context conditions. An event context condition ecc is evaluated for an event instance with respect to its context using the function $EV_{ecc}(ecc, \hat{e}) = f(EC(\hat{e}, r))$. ◇

When shared functionalities are discovered in the first matching phase, the user needs to define the necessary attribute and event context conditions in order to dissolve the assignment problem. Assume that all event instances of the incident event log have the attribute *role* and that the set of available comparison operators $O = \{equals, contains, startswith\}$. A context condition to identify if an event instance of the event class "CI" belongs to the activity "Investigation and diagnosis" could be written as ('role', 'equals', '2nd level'). To identify the cases when an instance of the event class "Group" belongs to the activity "Incident logging" we can create the event context condition (!◊('Group'), 'before'). The formula !◊('Group') is an LTL formula that stands for "eventually event 'Group' occurs". LTL has been chosen as it gives good flexibility for defining temporal conditions. In order to check the LTL statements on the event log we use the functionality of the ProM LTL Checker plug–in as introduced by Aalst et al. [124]. Note that this requires a particular syntax to distinguish between event labels and attributes. For a complete reference of the LTL Checker plug–in see [30].

It is also possible that multiple conditions need to match in order to map an event instance to an activity. Hence, we define a context condition c as the conjunction of attribute conditions and event context conditions as presented in Definition 10.

Definition 10 (Context condition). *A context condition* c *is a tuple* $c = (AC', ECC')$ *with* $AC' \subseteq AC$ *and* $ECC' \subseteq ECC$. *The set of all context conditions is denoted as* C. *The evaluation function checks whether all defined conditions hold for an event instance. It is defined as*

$$EV(c, \hat{e}) = \begin{cases} true & \forall\, ac \in AC' : EV_{ac}(ac, \hat{e}) = true \wedge \\ & \forall\, ecc \in ECC' : EV_{ecc}(ecc, \hat{e}) = true \\ false & otherwise \end{cases}$$

◇

Having defined the context conditions, we introduce an event class to activity mapping EAM based on event classes and conditions that have to be fulfilled for a corresponding event instance in order to be mapped to a specific life cycle transition of an activity.

Definition 11 (Event class to activity life cycle mapping). *An event to activity mapping is a relation* $LTEM \subseteq LTE \times C$, *which relates an event class to a life cycle transition of an activity based on a context condition.* ◇

Definition 11 gives the mapping between activities from a process model and event classes found in an event log. For our the event classes "CI" and "Group" of our incident process example, the event class to activity mapping could be defined by a domain expert as shown in Table 16. Note that the mappings, in general, should be non-

overlapping as there is no ordering in which the rules are tested and all matching relations will be taken into account in the next phase.

Table 16: Example event class to activity mapping for the event classes "CI" and "Group" .

$e \in E$	$a \in A$	$lt \in LT$	$c \in C$
CI	Incident closure	ϕ	({}, {(◇('Solution'), 'before')}
CI	Initial diagnosis	ϕ	({('role', 'equals', '1st level')}, {(!◇('Solution'), 'before')})
CI	Investigation & diagnosis	ϕ	({('role', 'equals', '2nd level')}, {})
CI	Security incident handling	ϕ	({('role', 'equals', 'Security')}, {})
Group	Incident logging	ϕ	({}, {(!◇('Group'), 'before')}
Group	Functional escalation	ϕ	({('role', 'equals', '1st level')}, {(!◇('Group'), 'before')})
Group	Investigation & diagnosis	ϕ	({('role', 'equals', '2nd level')}, {})
Group	Security incident handling	ϕ	({('role', 'equals', 'Security')}, {})

In order to give some examples for the context–sensitive removal of events or whole traces, consider the case where for each activity of the incident process model there is a protocol event containing a description of the concrete steps taken by the executing process participant. An analyst may decide to only keep the protocol events for the resolution of the incident, i.e., those that occurred after the solution was found and entered into the system. The mapping for this example is specified as follows: {('Protocol', 'REMOVE_EVENT', ϕ, ({}, {(!◇('Solution'), 'before')})), ('Protocol', 'Resolution and recovery', ϕ, ({}, {(◇('Solution'), 'before')}))}. Concerning the removal of complete traces, one might be interested in analyzing only incidents that do not concern security issues. To achieve this, we are required to remove all cases that contain security issues. These cases can be identified by event instances of the event class "CI" that have been created by a member of the role "Security". The following mapping tuple can be used to remove cases that contain such event instances: ('CI', 'REMOVE_TRACE', ({('role', 'equals', 'Security')}, {})).

Having defined the basic type–level mapping aspects to fulfill the given requirements, we now turn to two special cases that we encountered in our case studies that can potentially be captured with the previously defined concepts but only with higher manual efforts. So far, we have been looking at cases where an event class refers to a subset of the given activities and that reference may depend on one or more conditions. A special case is when an event class potentially belongs to every activity in the process model. One example for this are protocol events, which signal that a process participant documented what they did. Another example we encountered in practice are status events, which show if somebody is currently working on a case or if it is on hold. These events show the life cycle of the individual

activities and can for instance be used to calculate idle times within the execution of an activity. Such events typically belong to the activity that is currently executed or about to start. In order to achieve such a dynamic mapping, we introduce a special activity place holder that can be used in the LTEM mapping definition. This place holder is called *CLOSEST_ACTIVITY* and is processed after all other mappings. It signals the mapping algorithm to assign an event instance to the activity to which its closest neighbor is assigned to. The distance for the determination of the closest event is measured over the time stamps of the event instances.

While the conditions introduced in Definition 10 allow to mix attribute and context conditions, practical settings often require to assess whether a certain event has happened before and fulfills a certain attribute condition. Consider the scenario where the role attribute is not present in our example event log from Table 15. The mapping of event instance from the class "CI" in the incident management process partly depends on the role and on the fact whether an event instance of the class "Solution" has been seen before. With a close look at the given event log, one can see that the mapping can also be done without the role attribute by taking into account the value set in the previous event of the event class "Group". Whenever this value starts with "1st", the role that performed the subsequent steps is the first level role. If it starts with "2nd" or "Security", either a member of the second level or of the security role is responsible for the upcoming event instances.

In order to easily capture such dependencies, we introduce global event attributes that are added to each event instance. The values of these attributes are updated by the attribute values of occurrences of event instances of specific classes that are defined in the global attribute relation $GATR \subseteq \mathsf{E} \times ATE$. Assuming that the event instances of the class "Group" have an attribute called "value" that stores the group that has been set, the tuple for this example would be specified as ("Group", "value"). An attribute with the name "Group" is added to all event instances and its value is set to the last value provided by an event instance of class "Group", or the value will be empty if no such event instance occurred before. In this way, an attribute condition can be used to evaluate the current group when assessing the matching activity for an event of the event class "CI".

3.3.3 *Transformation of the Event Log*

Having defined the relations between event classes and activities on the type level, we can turn to the instance level. The goal of the phase described in this section is to transform the event log so that each event instance is mapped to its corresponding activity. Thus, we are mapping the instance level of the events to the type level of the activ-

ities. This is a necessary preprocessing step for the next phase, which deals with the clustering of event instances to activity instances. Definition 12 specifies the relation $\text{LT}\hat{\text{E}}$, which maps event instances to the life cycle transitions of the activities declared in the relation LTEM, for which all defined context conditions hold.

Definition 12 (Event instance to activity mapping). *Relation* $\text{LT}\hat{\text{E}} \subseteq \hat{\text{E}} \times \text{A} \times \text{LT}$ *defines the mapping of event instances to life cycle transitions of activity types, for which* $(\hat{e}, a, lt) \in \text{LT}\hat{\text{E}} \implies \exists\, (e, a, lt, c) \in \text{LTEM} : e = \#_{class}(\hat{e}),\ \text{Ev}(c, \hat{e}) = \textit{true}.$ ◇

Note that $\text{LT}\hat{\text{E}}$ is a relation that allows for multiple mappings of an event instance to different activities or activity life cycle transitions. This is a reflection of Requirement R4, which describes that there can be event instances that relate to multiple activities due to the use of shared functionalities. Consider the following example taken from one of our industry case studies: When an incident needs to be resolved with a change in the IT infrastructure, one has to document necessary steps as well as a back-out plan for the case something goes wrong during implementation. When the supporting IT system only reserves one field for these two texts, e.g. the field "Change Protocol", it will also only save one event for the change of this field. If the process model distinguishes the writing of the plan and back-out plan in two activities, there is one event instance representing two activities. Consider the event instance $\hat{e}_1$ with $\#_{class}(\hat{e}_1) =$ *'Change Protocol'* and $\#_{value}(\hat{e}_1) =$ *'Necessary steps: First do... then... If it fails, the back-out is done as follows:...'*. We know that for this case a protocol event always entails the writing of the plan. By defining conditions over the content of the protocol attached to the event instance as attribute "value", one can find out whether also the activity for writing the back-out plan has been executed, e.g., by searching for keywords such as "back-out". In this case, the relation LTEM contains multiple tuples for the event class "Change Protocol" that have matching context conditions for $\hat{e}_1$. These tuples are: {('Change Protocol', 'Document change plan', ϕ, ({}, {})), ('Change Protocol', 'Document back-out plan', ϕ, ({('value', contains, 'back-out')}, {}))}.

As stated before, the goal of this phase is to transform the event log such that each event instance is mapped to its corresponding activity. Algorithm 1 depicts the general algorithm for the log transformation. We iterate over the traces in a log and check for each event instance, for which tuples in LTEM all conditions hold. For each trace t, we create a new transformed trace t', which is added to the transformed event log L'. For an event instance $\hat{e}$ there may be multiple tuples in LTEM with matching conditions due to shared functionalities, as previously explained. All found tuples for an event instance $\hat{e}$ are added to the $\text{LT}\hat{\text{E}}$ relation (see line 9). For each of these tuples, the event instance $\hat{e}$ is copied and inserted into the transformed trace t' as new event instance $\hat{e}'$ (see line 12-18). The event class of $\hat{e}'$ is al-

Algorithmus 1 : Transform event log with $LT\hat{E}$.

```
1: transformLog(EventLog L, Relation LTEM)
2: EventLog L' := ∅
3: for all t ∈ L do
4:    Trace t' := ⟨⟩
5:    for all ê ∈ t do
6:       Set LTÊ := ∅
7:       for all (e, a, lt, c) ∈ LTEM do
8:          if e = #class(ê), Ev(c, ê) = true then
9:             LTÊ := LTÊ ∪ {(ê, a, lt)}
10:         end if
11:      end for
12:      for all (ê, a, lt) ∈ LTÊ do
13:         ê' := copy(ê)
14:         #class(ê') := a
15:         #source(ê') := #class(ê)
16:         #transition(ê') := lt
17:         t' := t' | ê'
18:      end for
19:   end for
20:   L' := L' ∪ {t'}
21: end for
22: return L'
```

tered to reflect the matching activity and the corresponding life cycle transition is set as additional attribute. In case multiple mappings match for an event instance $\hat{e}$ — as in the protocol example above — we duplicate $\hat{e}$ as many times as needed. In order to keep the relation to the original event class, we introduce a new event attribute $\#_{source}(\hat{e}')$, which contains the original event class. We refer to the attribute $\#_{source}(\hat{e}')$ as source event class. The result of this phase is a preprocessed event log where all event instances are assigned to their corresponding activity and, if specified in the relation LTEM, to the matching life cycle transition.

3.3.4 *Clustering of Event Instances to Activity Instances*

The previous section described the first transformation of the event log. In this transformation phase the event instances are mapped to activities on type level and, if known, to the corresponding life cycle transitions. Building on this mapping, the goal of the phase described in this section is to derive the final event log with a mapping from event instances to activity instances and life cycle transition instances. As described in Section 3.2, activity instances are not known without the connection of event log and process model. Having an event log where the event instances are mapped to their activities on type level, i.e., having the relation $LT\hat{E}$, activity instances can be derived using

clustering techniques. From the mapping to activity instances, the start, complete and execute life cycle transitions can be matched automatically. All other life cycle transitions have been assigned before using the user defined mappings from the relation LTEM.

The algorithm we propose for the clustering of events to activity instances is grounded on a tree-based incremental clustering algorithm known from classical data mining (cf. incremental clustering in [145]). For every activity in the process model the clustering forms a tree with the event instances as leaves and the activity instances on the next higher level. At the start of the clustering, all trees contain only an empty root node. The event instances are incrementally inserted into the tree. Updating the tree is done by finding the right place for a new leaf. This may cause restructuring of the affected part of the tree. Definition 13 formally introduces the activity instance tree, which essentially is a set of activity instances for a particular activity. Each activity instance is represented by a set of event instances. Hence, the activity instance tree clusters together those event instances that belong to the same activity instance.

Definition 13 (Activity instance tree). $\hat{AT}_i^a = \{\hat{AC}_1^a, \ldots, \hat{AC}_j^a\}$ *is an activity instance tree for the activity instances of activity* $a \in A$ *in trace* $t_i \in L$*, with* $i \in 1..|L|$*. The index* $j \in \mathbb{N}$ *symbols the number of activity instances of activity* a *in trace* t_i*.* $\hat{AC}_k^a = \{\hat{e}_1, \ldots, \hat{e}_l\}$*, with* $k \in 1..j$ *and* $l \in \mathbb{N}$*, is a set of event instances belonging to the activity instance* $\hat{a}_k$*.* $\hat{AC}_k^a$ *is also referred to as an activity instance cluster. The set of all activity instance clusters is referred to as* $\hat{AC}$*.* ◇

There is one activity instance tree $\hat{AT}_i^a$ for each activity $a \in A$ in each trace t_i, capturing all activity instances of a in trace $t_i \in L$. Algorithm 2 specifies the construction of activity instance trees in more detail. Algorithm 2 takes the transformed event log L' from the previous phase and iterates over all traces in the log. For each trace, the activity instance trees for all activities are initialized as empty sets (see line 4). We then iterate over all event instances in a given trace. For each event instance $\hat{e}$, we check whether the corresponding activity instance tree is still empty or not. The corresponding activity instance tree can be easily identified over the event class, because the previous phase already assigned the activity names as event classes. If the activity instance tree for the event instance at hand is still empty, the event instance is simply added as a member of a new set, i.e., a new activity instance cluster. In case there are already activity instance clusters in the activity instance tree, the algorithm determines the best host cluster for the event instance. The best host cluster for a new event is the cluster of event instances that contains the event instance with the minimal distance to $\hat{e}$. This distance between two event instances can be expressed by a distance function $dist : \hat{E} \times \hat{E} \to \mathcal{R}$. The distance function can use different attributes of the event instances or other information contained in the event log to calculate a distance. For

Algorithmus 2 : Clustering events to activity instances.

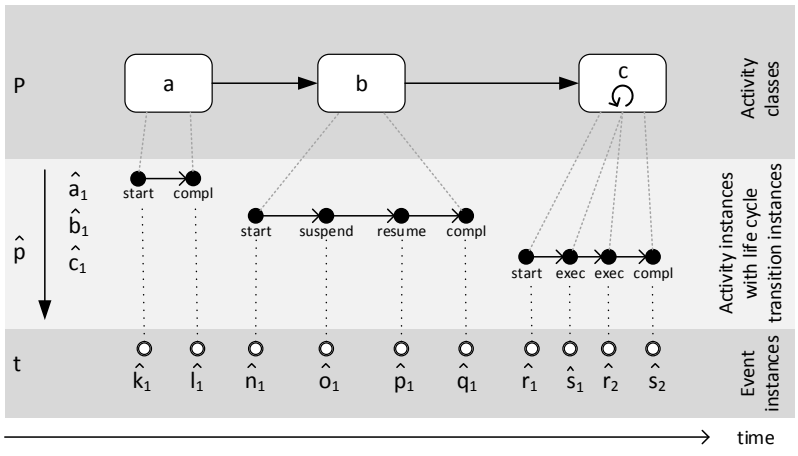
cluster(**Log** L'
for all $t_i \in L'$ **do**
  for all $a \in A$ **do**
    Set $\hat{AT}_i^a = \emptyset$
  end for
  for all $\hat{e} \in t_i$ **do**
    $a = \#_{class}(\hat{e})$
    if $\hat{AT}_i^a == \emptyset$ **then**
      $\hat{AC}_1 = \{\hat{e}\}$
      $\hat{AT}_i^a = \left\{\hat{AC}_1\right\}$
    else
      $distance_{min} = \infty$
      for all $\hat{AC}_j \in \hat{AT}_i^a$ **do**
        for all $\hat{e}_k \in \hat{AC}_j$ **do**
          if $dist(\hat{e}, \hat{e}_k) < d_{min}$ **then**
            $\hat{AC}_{host} = \hat{AC}_j$
            $distance_{min} = dist(\hat{e}, \hat{e}_k)$
          end if
        end for
      end for
      $instanceBorderFound = false$
      for all $\hat{e}_k \in \hat{AC}_j$ **do**
        if $checkInstanceBorder(\hat{e}, \hat{e}_k) == true$ **then**
          $instanceBorderFound = true$
          break
        end if
      end for
      if $instanceBorderFound == false$ **then**
        $\hat{AC}_{host} = \hat{AC}_{host} \cup \{\hat{e}\}$
      else
        $\hat{AC}_{|\hat{AT}_i^a|+1} = \{\hat{e}\}$
        $\hat{AT}_i^a = \hat{AT}_i^a \cup \left\{\hat{AC}_{|\hat{AT}_i^a|+1}\right\}$
        $restructure(\hat{AT}_i^a)$
      end if
    end if
  end for
  for all $a \in A$ **do**
    for all $\hat{AC}_j \in \hat{AT}_i^a$ **do**
      $merge(\hat{AC}_j)$
    end for
  end for
end for

example it may use the timestamps, the number of event instances in between, the calculated distance between other event attributes or a combination of these. In case we can exclude concurrent executions of the same activity and do not include specific event attributes, the best host will always be the last inserted event of the same activity as events are sorted chronologically in the event log. This event can easily be identified using the order relation $>^{\hat{e}}$. The last event instance in an activity instance cluster $\hat{AC}_j^a$ is the event instance $\hat{e}_l \in \hat{AC}_j^a$ for which holds $\forall\, \hat{e}_m \in \hat{AC}_j^a : \hat{e}_m \neq \hat{e}_l \implies \hat{e}_l >^{\hat{e}} \hat{e}_m$.

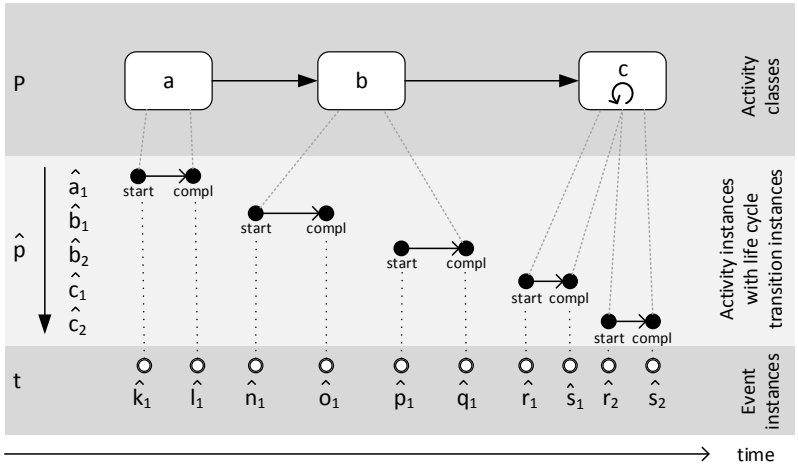
For now, we only explained how to find the cluster to which an event instance may be assigned, but we did not explain when a new cluster is created. One activity instance tree may contain multiple activity instance clusters, as there might be multiple activity instances for one activity in a process instance, i.e., in a loop. Figure 28 depicts different results for the activity instance clustering of event instance for which the type–level mapping is already given. In Figure 28a all event instances that are assigned to an activity on type level are clustered into a single activity instance. The clustering in Figure 28b yields always clusters of size two, i.e., two activity instances for both activities b and c. In Figure 28c the event instances of activity b are clustered to one activity instance while there are two activity instances for activity c. Although the clustering in Figure 28c seems to be the one that is most likely correct, we cannot tell if it is. As stated in Requirement R9, the execution may be nonconforming to the process model. If this is the case, also the clusterings in Figure 28a and Figure 28b may be the correct mappings. In order to specify the correct clustering, additional knowledge needs to be encoded. We need to define the border between events belonging to two or more instances of the same activity. We therefore introduce the notion of instance border conditions.

Definition 14 (Instance border condition, extended event class to activity life cycle mapping). *An instance border condition defines whether an event instance belongs to an activity instance cluster* $\hat{AC}_i^a$ *or if it belongs to another activity instance cluster* $\hat{AC}_{i+1}^a$. *It is defined as a boolean function* $bc : \hat{E} \times \hat{E} \to \{true, false\}$. *The set of all border conditions is* BC. *Each tuple in* LTEM *is extended by a border condition function such that* $LTEM^* \subseteq LTEM \times BC$. *An event instance* $\hat{e}_j$ *cannot belong to an activity instance cluster* $\hat{AC}_i^a$, *if it holds* $\exists\, \hat{e}_k \in \hat{AC}_i^a, bc(\hat{e}_j, \hat{e}_k) = true$ *for any activity instance border condition* bc *that is part of a matching tuple in* $LTEM^*$. ◇

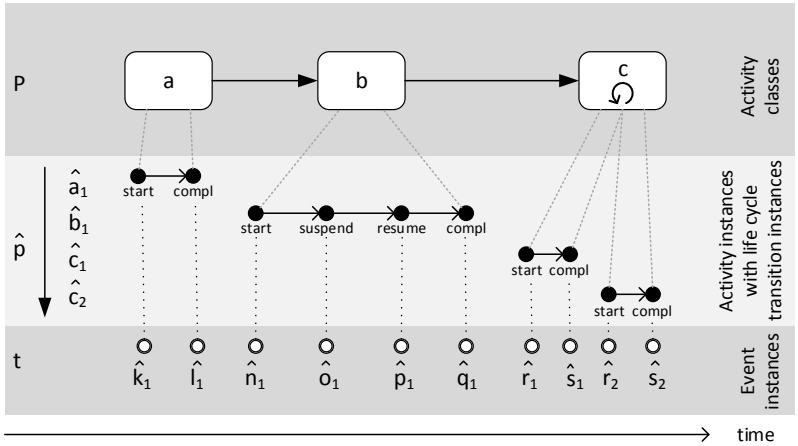
Definition 14 introduces the notion of activity instance borders. Each tuple in LTEM is extended with an instance border condition, building the extended event to activity life cycle mapping $LTEM^*$. Note that this is a manual step that needs to be performed by an analyst. Yet, from our practical experience it is often sufficient to globally



(a)



(b)



(c)

Figure 28: Different results for clustering activity instances.

define one instance border condition that is used in all mapping tuples. Thus, this step does not necessarily require a large amount of manual effort.

Instance border definitions relate to two levels: intra and inter activity structure. Concerning the *intra activity* structure, we have to decide whether there are loops in activities on the sub–activity or life cycle level or not. In case the sub–activity or life cycle level is unknown, an analyst has to make an assumption about looping behavior on sub–activity or life cycle level. The *inter activity* level refers to loops and repetition on activity level. While the process model might not contain loops on inter activity level, this does not imply that there are no loops on the inter activity level in the execution if the execution is not strictly bound to the process model. In case we assume that there are no loops on intra activity level, any repetition needs to be lifted to the activity level. That is, an activity instance border is marked by the repetition of source events from the same event class, i.e., the repetition of a source event class signals that a new activity instance has started. Thus, for example two protocol events could indicate rework and therefore two instances of the corresponding activity.

Using recurring source event classes as instance border definition works only if there are no loops in the assumed sub–activity model. If there are loops on the intra and inter activity level, multiple event instances from the same event class might belong to one activity instance. A typical example for this is a loop over the order items in an order list where different activities like "Choose supplier" have to be performed for each order item and are modeled on activity level. The activity "Choose supplier" might contain different sub–activities that have to be executed for each supplier, like, e.g., "Check prices". Thus, we have a loop on activity level and a loop on sub–activity level. In order to find the correct instance borders, we need to extend the instance border definition to also use different business objects, e.g., the order line, as instance border markings. Thus, instance borders can be defined over any attributes attached to an event.

If necessary attributes are not available or if it is not possible to make statements about the assumed sub–activity or life cycle model, we need to use *heuristics* in order to be able to identify different activity instances. Similar to the approaches of Li et al. [75] and Günther et al. [57, 59], a first heuristic for an instance border can be defined based on a threshold for the maximum distance between two events that belong to one activity instance. While previous works only focus on a very narrow and short distance based on the number of events in between two events, we extend this definition using also the time perspective, i.e., defining how long the time frame between two event instances of the same activity instance can be. For example one might limit the time distance between two event instances of the same activity instance, e.g., two edit events for a protocol belong to

different activity instances if there are more than 24 hours between them. Using a maximal number of events that are allowed to occur between two events of the same activity instance still makes sense as a heuristic from our practical experience. Yet, it needs to be specified by the user and should also allow larger distances when there are many events assigned to single activities and concurrency cannot be precluded. In a similar manner, another heuristic can be built on the assumption of a maximum number of event instances that belong to one activity instance. Here, a domain expert has to estimate how many sub–activities or life cycle transitions are approximately executed per activity instance. This is simple, if we can exclude loops on sub–activity level, but might be difficult otherwise. In the former case, one would limit the maximum number of event instances to the number of assigned event classes for an activity.
Having defined the instance border conditions, Algorithm 2 uses these definitions when inserting a new event instance into an existing activity instance cluster. For all events that belong to the same activity instance as the host event, the algorithm checks that these do not fulfill any border condition in combination with the event that has to be inserted into the cluster (see line 21-27 of Algorithm 2). Note that in this step we only check if an instance border exists. We do not yet know where this border is if a border is found. The new activity instance might already have started before the event is observed that signals an instance border. If no instance border is found, the event instance is added to the host set. When an activity instance border is found, a new set is created for the additional activity instance and the event instance $\hat{e}$ is added to this set. As said before, we do not know whether the current event instance is the start of a new event instance. It can be the case that the other event instances that have been previously assigned to another activity instance, may actually also belong to the newly created activity instance, e.g., if their distance is closer to $\hat{e}$ than to any of the other event instances. Therefore, a goodness measure g is introduced which enables us to make relative comparisons between different cluster possibilities.

Definition 15 (Goodness measure). *A goodness measure is a function $g : \hat{AT} \to \mathcal{R}$, which calculates the goodness as a real number for an activity instance tree. It is defined as follows:*

$$g\left(\hat{AT}_i^a\right) = \begin{cases} \sum_{\substack{(\hat{e}_m,\hat{e}_n)\in \hat{AC}_1^a \times \hat{AC}_1^a, \\ \hat{e}_m \neq \hat{e}_n}} \mathrm{dist}\left(\hat{e}_m, \hat{e}_n\right) & \text{if } \hat{AT}_i^a = \left\{\hat{AC}_1^a\right\} \\ \sum_{k=1}^{k=|\hat{AT}_i^a|} g\left(\hat{AC}_k^a\right) & \text{otherwise.} \end{cases}$$

◇

The function g takes an activity instance tree and sums up the goodness values for all contained activity instance clusters. The goodness value of a particular activity instance cluster is calculated by summing up the distances between all pairs of events in the cluster. Note

that the goodness function can of course be varied. For example, if only the time and number of events that occur between two events, it is enough to calculate the distances between adjacent event instances. Using the goodness value one can compare different clustering results with each other and choose the best. Smaller values for g signal better clusters as the events within the cluster are closer to each other.

The algorithm starts by adding the new event under a new activity instance. Next, event after event are moved from the host activity instance to the new activity instance. At each point in this iteration the goodness measures for the two activity instances and the root are calculated. It is checked that no instance border condition is fulfilled between any two events of an activity instance. Otherwise the cluster combination is discarded. If we exclude concurrent executions of the same activity, it is sufficient to move event after event from the host cluster to the new cluster starting with the last inserted event and ending before the event is moved which fulfills the border condition. Otherwise, all combinations have to be tested which leads to exponential effort ($\mathcal{O}(\frac{2^n}{2})$). In the end, the clustering with the lowest goodness value is chosen as the optimal cluster and gives us the border between the two activity instances.

We will explain the basic algorithm using an example. Figure 29 shows an example event sequence. The mapping LTEM for this example is defined so that every event class x_i maps to an activity x and every event class y_i maps to an activity y. Looking at the instance border conditions, we define that there are no loops on sub–activity or life cycle level. We further define that there are no loops and no parallel executions of instances of the same activity.

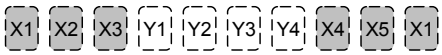


Figure 29: Event sequence example.

Figure 30 shows the clustering steps for activity x. Events are added to the cluster as long as they do not fulfill the defined border conditions. When adding the second instance of event x_1, it is recognized that there is already an instance of x_1 in the current cluster. Thus, a border condition is fulfilled and a new cluster for a new instance has to be created (Figure 30 (2)).

Looking at the clustering example in Figure 30, the insertion of the second x_1 at first results in two clusters with $g_{C1} = 8$ and $g_{C2} = 0$. Next, the algorithm moves event after event from the first cluster to the second cluster until he finds the optimal solution. In the example, the optimal result with $g_C = 4$ contains the two clusters as shown in Figure 29 (4).

Having the activity instance clusters for each activity, we can assign the life cycle transitions to those event instances that have been

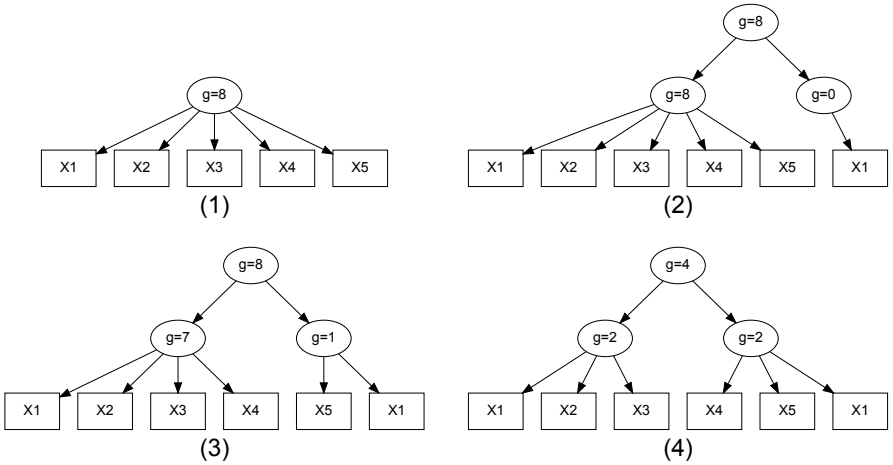


Figure 30: Clustering events of activity X to activity instances using tree structures.

mapped to the placeholder transition ϕ. If the first event instance in an activity instance cluster $\hat{AC}_j^a$ is assigned to the placeholder transition ϕ, it is mapped to the start transition. The first event instance $\hat{e}_l \in \hat{AC}_j^a$ is identified by checking that $\forall\, \hat{e}_m \in \hat{AC}_j^a : \hat{e}_m \neq \hat{e}_l \implies \hat{e}_m >^{\hat{e}} \hat{e}_l$. In a similar fashion, the last activity instance of an activity instance cluster is mapped to the complete transition. Any other event instance that is mapped to the placeholder transition ϕ is assigned to the introduced *execute* life cycle transition. These event instances can then be easily removed, if they are not required for the intended analysis.

In case the event log needs to be preprocessed for an analysis technique that requires a one-to-one mapping between event instances and activity instances (Requirement R1), the user can decide whether only the first or the last event instance of an activity instance cluster should be kept. All other event instances are deleted in that case.

In order to address the requirement of hierarchical mappings to support zoom-in functionalities in process discovery techniques (Requirement R8), it is furthermore possible to save each activity instance tree as a separate event log that is linked to each of its activity instances in the original event log. Therefore, each activity instance clustered is saved as a trace of the corresponding activity instance event log. The source attribute of each event instance is used as its event class. Using multiple mappings LTEM, hierarchies with arbitrary numbers of levels can be built by iteratively applying the base approach.

3.4 SUMMARY

This chapter laid the groundwork of this entire thesis by formalizing the matching problem and by defining the required concepts neces-

sary for the mapping of activities and events. All of the previously defined requirements were taken into account for the formalization and the subsequent derivation of the base approach. Coming from the formalization of the mapping, the four phases of the base approach are introduced. For each phase the inputs and outputs are defined.

The approach starts by mapping activities and events on the type level in the first phase. Event classes are mapped to their corresponding activities. Additionally, event classes can be assigned to activity life cycle transitions. However, the assignment of start and complete transitions does not need to be specified manually. Next, context-dependent mappings are created to be able to match events that stem from shared functionalities. Having the context–sensitive mapping definitions, the third phase transforms the log event log by renaming the event instances with their corresponding activity label. The resulting event log is further processed in the last phase by clustering event instances that belong to the same activity into activity instances. For this purpose, the concept of activity instance borders is introduced in order to distinguish between events of different activity instances stemming from a designed loop or from unwanted rework. A tree-based incremental clustering algorithm with a flexible distance measure is used for the clustering of event instances to activity instances. Finally, the start and complete life cycle transitions are automatically derived and the log is transformed so that all event instances are related to their corresponding instances of activity life cycle transitions.

4

APPROACH BASED ON LOG REPLAY

This chapter introduces a semiautomatic approach for the mapping of events to given activities in a process model based on the building of a constraint satisfaction problem from replaying the log on the model. We will refer to the approach described in this chapter as the replay approach. The work presented in this chapter has been published in [9]. The replay approach builds on the base approach, which has been introduced in Chapter 3, and provides a semiautomatic means for the matching of events and activities on type level. This step is a purely manual activity in the base approach. We start by listing the assumptions of the approach and the requirements it fulfills in Section 4.1. The subsequent sections introduce the concepts that distinguish the replay approach from the base approach.

4.1 REQUIREMENTS AND ASSUMPTIONS

In order to automate the matching of events and activities on type level using a log replay technique, we have to make additional assumptions to the ones stated for the base approach. In contrast to the base approach, the approach presented in this chapter demands a process model as Petri net that fulfills the following assumptions: It has to have an initial and, at least, one final marking. Furthermore, the Petri net has to be bounded and every labelled transition has to be part of at least one sound firing sequence. This assumption is similar to the relaxed soundness defined by Dehnert and Rittgen [35], but does not require the Petri net to be a workflow net.

The replay approach requires the type–level relation of activities and events (AE) to be a one-to-one relation. Moreover, also the instance level relation ÂÊ needs to be a one-to-one relation. Strictly speaking, the event log has to be on the same abstraction level as the process model. Thus, the replay approach fulfills Requirement R1 (1:1 matching to activities) but does not fulfill Requirement R2 (Different abstraction levels), Requirement R4 (Shared functionalities), and Requirement R8 (Hierarchical matching).

Regarding the life cycle of activities, we assume that all activity life cycles only have two states and one complete transition between them. Events in the event log refer to this complete transition and thus, no further specific mapping has to be done. Hence, Requirement R7 is fulfilled by the replay approach as a consequence of this assumption. Additional events, which do not belong to any activity, need to be filtered out before and activities that are not recorded need to be removed from the model or declared silent. Hence, Requirement R6

and Requirement R5 are not handled by the approach presented in this chapter.

The recorded execution in the event log has to be mainly conforming to the given process model. That is, the majority of traces in the event log need to fully conform to the process model. We thereby do not exclude non-conforming behavior in the event log and thus, still fulfill Requirement R9. Yet, we need to put this extended assumption on the conformance of complete traces into place in order to make the replay approach work. From our practical experience, it is often possible to reach higher levels of conformance by filtering the event log for the most frequently occurring trace variants. When such a filtering is applied, one has to ensure that all event classes are still contained in the filtered event log.

4.2 OVERVIEW OF THE REPLAY APPROACH

As explained before, the replay approach introduces a semiautomatic way to match events and activities on type level. Figure 31 depicts the main steps of the replay approach. The replay approach builds on the base approach and, therefore, uses the same procedure as the base approach (see Figure 27). The difference lies in the implementation of the first step, the matching of activities and events on type level. The replay approach introduces three sub–activities:

1. Reduce potential set of event–activity mappings,
2. Select correct mapping, and
3. Assign life cycle transitions.

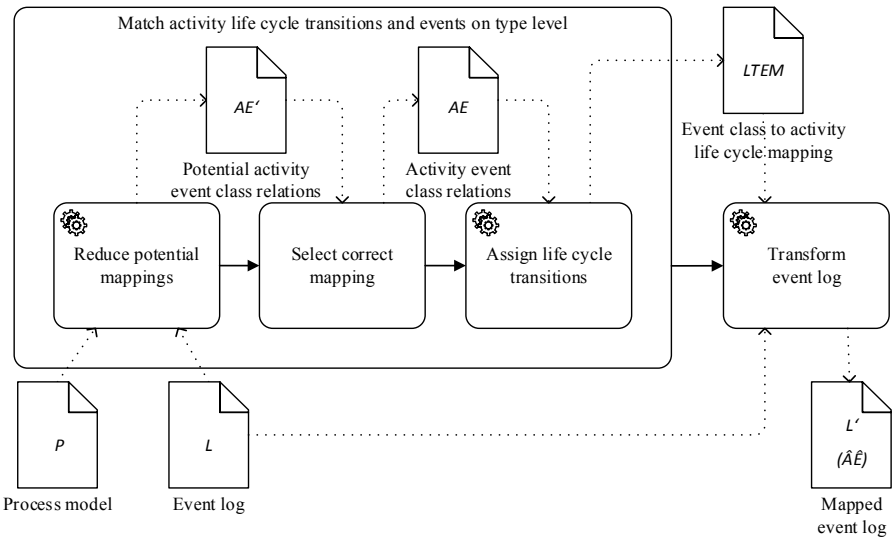


Figure 31: Steps for matching of events and activities using the replay approach.

The first sub–activity, "Reduce potential mappings", is an automated step that builds and solves a constraint satisfaction problem to reduce the number of possible mappings between activities and events. The result of this step is a set of potential activity event relations (AE'). During the second sub–activity, "Select correct mapping", the analyst is guided to select the correct mapping from the derived potential mapping, resulting in the activity event relation AE. In the last sub–activity of the type–level matching of events and activities, the "complete" life cycle transitions are automatically assigned to each tuple in AE, leading to the relation LTE. The further steps are identical to the steps described in the base approach. Yet, the clustering of event instances to activity instances is not necessary in the one-to-one setting. Thus, the final event log is already achieved by the activity "Transform event log".

The main differences to the base approach lie in the first two sub–activities of the matching of activities and events on type level, i.e., in "Reduce potential mappings" and "Select correct mapping". We will elaborate on these two steps in the following sections.

4.3 REDUCTION OF POTENTIAL EVENT–ACTIVITY MAPPINGS

The first step of our approach deals with the definition of a constraint satisfaction problem (CSP) that is used to restrain the possible mappings of events and activities. We refer to 2.2 for the definition of a CSP. To build the CSP, first, the activities and event classes need to be mapped to the set of variables and their domains. The bijective function $var : E \to \mathcal{X}$ assigns each event label to a variable with the natural numbers $1..|A|$ as domain. The activities are projected onto natural numbers by the bijective function $val : A \to 1..|A|$, which assigns each activity a natural number in the range from 1 to the number of activities.

Table 17 and Table 18 show the mapping var and the mapping val respectively for the order process example given in Section 2.3.1.

Table 17: Mapping var for the order process.

Event class $e \in E$	**Variable** $var(e) \in \mathcal{X}$
O_CHK	x_1
O_RCO	x_2
O_PRC	x_3
I_SM	x_4
P_SP	x_5
P_NOT	x_6
O_ARC	x_7

Table 18: Mapping val for the order process.

Activity $a \in A$	**Value** $val(a) \in 1..\|A\|$
Check order	1
Change order	2
Process order	3
Send invoice	4
Ship Products	5
Send notification	6
Archive order	7

Because we are looking at a one-to-one relationship between events and activities, we can specify a constraint capturing that no two activities can be mapped to the same event label. This constraint is defined as $c_1 \equiv \forall (x_i, x_j) \in X^2 : i \neq j \implies x_i \neq x_j$. It is available in most constraint solvers as the *allDifferent* constraint. With the variables, domains, and the *allDifferent* constraint defined, the solutions to the CSP reflect all possible mappings between events and activities, i.e., for n activities and event classes there are $n!$ solutions. For the example given in Table 5 and Table 6 in Section 2.3.1 these are $7! = 5040$ possible mappings. In the following, we present an approach to tackle this complexity by combining the information available in the log with the knowledge of the process model structure.

To generate more constraints and thus be able to reduce the number of possible mappings, the next step of the first phase is the replay of the event log in the process model. Because the event log can contain multiple traces that encode the same ordering of events, the log is preprocessed to extract all unique variants of traces. The tuple $V = \langle v_1, v_2, \ldots, v_k \rangle$ contains all variants $v_i \in L$, $i \in 1..k$ and the tuple $W = \langle w_1, w_2, \ldots, w_k \rangle$ holds the number of occurrences for each variant, e.g., the variant v_1 is contained w_1 times in the log L (see also Section 2.1.8). As the example log in Table 5 only contains unique traces, v_i refers to trace t_i, and $w_i = 1$ for any $i \in 1..k$ in the following examples.

As the relation between events and activities is unknown at this stage, the replay of one trace variant v_i is essentially a mapping of each trace variant v_i to all possible execution sequences that have the same length as v_i. Therefore, we define the relation $vpi \subseteq V \times \hat{P}$ such that $vpi = \{(v_i, \hat{p}_j) \mid v_i \in V, \hat{p}_j \in \hat{P}, |v_i| = |\hat{p}_j|\}$.

The relation vpi describes possible mappings between sequences of event labels and sequences of activities. In fact, we are interested in limiting the number of possible mappings to those that result in the highest number of traces with valid execution sequences, i.e., in the mapping that yields the *maximal conformance* when replaying the log with the mapped events.[1] Each tuple in vpi reflects a replay of a trace variant in the model. For easier explanation of the procedure, let us first assume that all traces in the log conform to the model. First, a constraint $cst_{i,j}$ is created for each tuple $(v_i, \hat{p}_j) \in vpi$. The constraint $cst_{i,j}$ reflects a mapping of event classes to activities by assigning each event class to the activity at the same position in the sequence. Hence, $cst_{i,j}$ has the form $\bigwedge_{k \in 1..|v_i|} var(a_k) = val(e_k)$. Note that there can be several paths in the model that have the same length as the trace variant. In case of conforming execution, we need to ensure that for each trace variant v_i one of these constraints holds, i.e., that one of the defined mappings allows a valid replay of the trace variant in

1 Conformance of event logs to process models can for example be measured using the approach introduced by Rozinat and Aalst in [100].

the model. Therefore, a constraint cst_i is formulated for each trace variant. The constraint cst_i has the form $\bigvee cst_{i,j}, \forall j : \exists (v_i, \hat{p}_j) \in vpi$.

Consider the variant $v_1 = \langle$ O_CHK, O_PRC, I_SM, P_SP, O_ARC $\rangle$. There are two execution sequences in the model that have the same length as v_1. These are $\hat{p}_1 = \langle$ Check order, Process order, Send invoice, Ship products, Archive order $\rangle$ and $\hat{p}_2 = \langle$ Check order, Process order, Ship products, Send invoice, Archive order $\rangle$. Hence, we first create the two constraints $c_{1,1} : x_1 = 1 \wedge x_3 = 3 \wedge x_4 = 4 \wedge x_5 = 5 \wedge x_7 = 7$ and $c_{1,2} : x_1 = 1 \wedge x_3 = 3 \wedge x_5 = 4 \wedge x_4 = 5 \wedge x_7 = 7$. Given the two constraints $c_{1,1}$ and $c_{1,2}$, the constraint $c_1 : c_{1,1} \vee c_{1,2}$ is derived. By adding the constraint c_1 to the CSP, we already fix the mappings of "Check order", "Ship products" and "Archive order" and thereby limit the possible mappings from $7! = 5040$ to $(7-3)! = 24$. Once the constraint c_2 for trace variant v_2 has been built in the same manner and added to the constraint satisfaction problem, the number of solutions satisfying both constraints is reduced to a single one, which is the mapping as specified in Table 6.
Yet, adding the constraint c_3 for trace variant v_3 results in a CSP that has no solution. This is due to the fact that v_3 is not conforming to the model. The CSP tries to satisfy all constraints and thus requires every trace variant to conform to the model. Therefore, to handle nonconforming traces, the CSP is reformulated as an optimization problem. The optimal solution to the problem is a mapping in which the maximum number of traces conforms to the model. It is important to note that we assume that there is a sufficient number of conforming traces in the log to be able to retrieve a correct mapping.

The constraint cst_i, which has been built for each trace variant, is therefore used to define a boolean variable $validVariant_i$ for each trace variant as follows:

$$validVariant_i = \begin{cases} 1 & cst_i = \text{true} \\ 0 & \text{otherwise.} \end{cases}$$

The variable $validVariant_i$ reflects whether trace variant v_i represents a valid execution sequence with the chosen mapping. Having defined the variable $validVariant_i$ for each trace variant, a new variable $validTraces \in 0..|L|$ is introduced that sums up all valid traces by multiplying the valid variants with the number of traces sharing the corresponding behavior:

$$validTraces = \sum_{i=1}^{|V|} validVariant_i \cdot w_i.$$

The variable $validTraces$ is set as the optimization goal that should be maximized when solving the CSP. This way, the CSP for the example can be solved with $validVariant_1 = 1$, $validVariant_2 = 1$ and $validVariant_3 = 0$, yielding $validTraces = 2$. The optimal solution

is again the correct mapping as shown in Table 6. Hence, the approach is able to deal with nonconforming traces in the log as requested by Requirement R9. Furthermore, this shows that it is not necessary to have a complete log containing all possible behavior in order to construct an unambiguous mapping. Note, however, that there are cases where it is not possible to reduce the number of solutions to a single solution. In the next section we discuss these cases and clarify how they can be handled.

4.4 SELECTION OF THE CORRECT TYPE–LEVEL MAPPING

The previous section introduced the approach for an automated type–level matching of event classes and activities. Still, there are cases for which no unambiguous mapping can be automatically derived. Basically, this is due to two common control flow constructs: choice and concurrency. Figure 32a and Figure 32c depict the simplest forms of these two constructs. While it is impossible to unambiguously derive a mapping for activities a and b in these two cases, it is possible for the cases depicted in Figure 32b and Figure 32d. This is due to the fact that for case (a) and case (c) the branches are equivalent in their behavior, while they are not in case (b) and case (d). For case (b) there are two possible trace variants. Yet, a single trace is enough to unambiguously determine a mapping between corresponding events in a log and the activities in the model if we assume that the available event classes are known. To give an example, we only need a trace with the two events corresponding to activities b and c, or a trace with the event corresponding to activity a. Regarding case (d), there are three possible trace variants. Still, two different traces are enough to unambiguously distinguish the activities from each other, because activity a is the only activity that can be first and last. Note that to reach an unambiguous mapping it is required to see at least one trace in which activity a was executed before, and another trace where it was executed after the parallel activities b and c. If this information is not contained in the log for any reason, a certain ambiguity remains in the automatically derived mapping.

Summing up, there are two main sources for ambiguities in the mapping. First, choices and parallel branches with identical behavior in the branches cause ambiguities. In this case, the number of the undistinguishable branches combinatorially increases the potential mappings. The second source for ambiguities is behavior that is possible in the model but not contained in the event log.

Ambiguous mappings, i.e., cases in which the CSP has multiple solutions, cannot be automatically resolved and require a domain expert to decide the mapping for the concerned events and activities. Nonetheless, this decision can be supported by the mapping approach. To aid the analyst with the disambiguation of multiple poten-

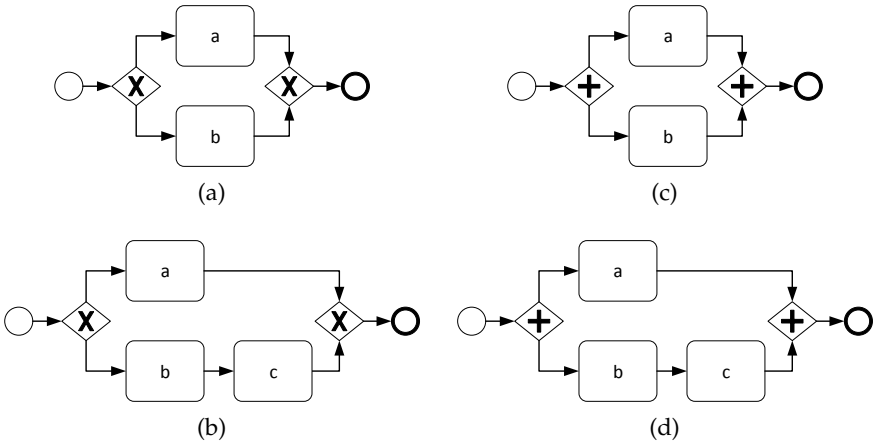


Figure 32: Control flow patterns for choice and concurrency with different impact on potential mappings.

tial mappings, we introduce a questioning approach. Our questioning approach is inspired by the work of La Rosa et al. [99], in which the user is guided through the configuration of a process model using a questionnaire. One event class and its activities that are potentially matching are presented to the analyst at one time. Once the analyst decides which of the candidate activities belongs to the event class, this mapping is converted into a new constraint that is added to the CSP. Consecutively, the CSP is solved again. In case there are still multiple solutions, the analyst is asked to make another decision for a different event label. This procedure is repeated until the CSP yields a single solution. Figure 33 shows the explained procedure for resolving ambiguous mappings in the light of the overall mapping approach. The goal is to pose as few questions to the analyst as possible.

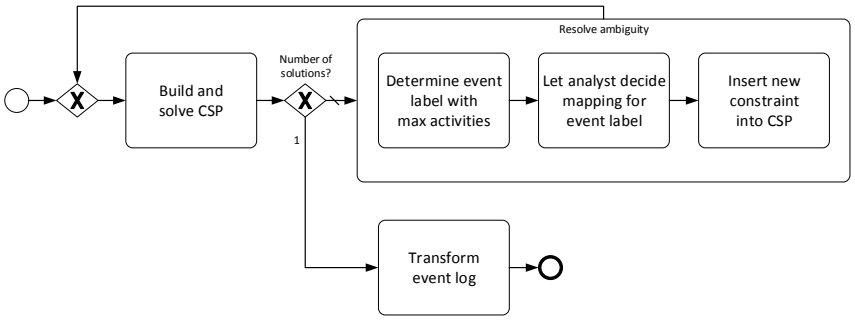


Figure 33: Detailed flow of the matching approach based on log replay.

To achieve this goal, we look into all solutions and choose the event label that is assigned to the maximal number of different activities. To illustrate this principle, consider the example trace $t_1 = \langle \hat{k}, \hat{l}, \hat{m} \rangle$. The events of which should be matched to the activities in the model in Figure 32d. Building and solving the CSP for this example leads to three solutions: $S = \{\langle x_1 = 1, x_2 = 2, x_3 = 3 \rangle, \langle x_1 = 2, x_2 = 1, x_3 = 3 \rangle, \langle x_1 = 2, x_2 = 3, x_3 = 1 \rangle\}$. The value 1, which corresponds to activity a

in this case, is assigned to all three variables, which correspond to the event classes “k”, “l” and “m”. Opposed to this, the other two values are only assigned to a subset of the three variables. By deciding the matching event for activity a, the CSP contains only one solution. Deciding the matching for any of the other two event labels results in a CSP with two possible solutions.

4.5 SUMMARY

In this chapter, we have introduced the replay approach, which builds on the base approach and adds a semiautomated approach for the type–level mapping. The presented approach replays the event log on the process model to constructs constraints that can be inserted into a constraint satisfaction problem to model the matching of events and activities on type level. In order to facilitate the replay, the approach relies on the assumption that both event log and process model are on the same abstraction level. The requirements Requirement R2 (Different abstraction levels), Requirement R4 (Shared functionalities), and Requirement R8 (Hierarchical matching) can therefore not be fulfilled. These shortcomings are traded in for automated support in the type–level mapping, which has been completely manual in the base approach. Since only one-to-one mappings between activities and events are supported, the step for defining context–sensitive mappings and the final clustering to activity instances can be omitted in the replay approach.

The basic rationale behind the replay approach is that once an event log is mapped to a process model, the traces in the event log shall be valid execution sequences of the process model. In order to handle nonconforming executions that are recorded in an event log (Requirement R9), the developed constraint satisfaction problem is transformed into an optimization problem that returns the mapping or mappings where the maximum number of traces are valid execution sequences of the process model. A questioning approach is introduced to guide the analyst through multiple potential mappings when the solution of the optimization problem does not provide a unique mapping. The questioning approach is designed to minimize the number of questions an analyst has to answer.

5

APPROACHES BASED ON BEHAVIORAL RELATIONS

This section introduces two approaches that build on behavioral relations to provide a semiautomated means to retrieve the type–level relations between activities in events. Namely, these approaches leverage behavioral profiles and Declare rules for the type–level matching. The approaches are based on our work published in [11] and [10]. Again, the approaches based on behavioral relations build on the base approach and require further assumptions on the input data. These assumptions are laid out in Section 5.1, together with the requirements that the approaches fulfill. Section 5.2 provides the general framework for the approaches based on behavioral relations. Section 5.3 and Section 5.4 introduce means to derive constraints using behavioral profiles and Declare rules respectively. In Section 5.5, we elaborate on specific cases where some constraints need relaxation. Section 5.6 and Section 5.7 explain the options for solving the derived constraint satisfaction problem and how the analyst is guided through the results.

5.1 REQUIREMENTS AND ASSUMPTIONS

In the same manner as for the replay approach, the approaches based on behavioral relations demand further assumptions on the input data than those declared for the base approach. Again, we assume a bounded Petri net with an initial and at least one final marking to be present. All transitions representing activities have to be part of at least one sound firing sequence. In contrast to the replay approach, the type–level relation of activities and events (AE) does not need to be a one-to-one relation. The approaches based on behavioral relations are able to handle event logs that are on a lower abstraction level than the provided process model. The type–level relation of activities and events (AE) is therefore a one-to-many relation. Thus, the approaches presented in this chapter do not only fulfill Requirement R1 (1:1 matching to activities), but also Requirement R2 (Different abstraction levels). Using the same techniques as introduced in the base approach, also Requirement R8 (Hierarchical matching) can be fulfilled. As only one-to-many relations are handled, Requirement R4 (Shared functionalities) cannot be served, because it requests the handling of many-to-one or even many-to-many relations. While many-to-one relations could be handled using a simple modification of the general approach, they are not considered as event logs are typically either on the same abstraction level or on a lower abstraction level, leading to either one-to-one, one-to-many or many-to-many relations.

In order to fulfill Requirement R7 (1:1 matching to life cycle transitions), the same approach is used as in the base approach. The user may define the mappings of specific life cycle transitions, while start and complete transitions are automatically assigned by the activity instance clustering algorithm.

As for the replay approach, Requirement R5 and Requirement R6 cannot be handled. Therefore, additional events need to be filtered out before and activities that are not recorded need to be removed from the model or declared silent.

Concerning Requirement R9, i.e., nonconforming execution, the assumption taken for the replay approach can be relaxed when using behavioral relations. The approaches based on behavioral relations do not require complete traces to be conforming. They can potentially also handle event logs where not a single trace completely conforms to the process model. Yet, the approach works best if there is no systematic misbehavior in the log, meaning that the relation of two events is systematically different to the relation of their corresponding activities. Take for instance two activities a_1 and a_2 that are in strict order to each other ($a_2 \rightsquigarrow a_1$). If the activity to event relations on type level are $\{(a_1, e_1), (a_2, e_2)\} \subseteq AE$, e_1 and e_2 should also be in strict order, i.e., $e_2 \rightsquigarrow e_1$, in the majority of traces where both events occur. A certain amount of misbehavior can be easily handled making the approach robust to nonsystematic nonconforming behavior. Using an optimization problem in a similar way as in the replay approach, this assumption can be further relaxed to also allow some relations in the event log to be systematically different than the corresponding relations in the process model. Yet, it comes with the price of performance degradation. As being said before, the conformance of an event log to a process model can often be improved by concentrating on the most frequent trace variants. Yet, it needs to be ensured that all event classes are contained in the event log used for the approaches described in this chapter.

5.2 GENERAL APPROACH BASED ON BEHAVIORAL RELATIONS

In a similar manner to the replay approach introduced in Chapter 4, the approaches based on behavioral relations detail the matching of activities and events on type level in a semiautomatic fashion. Figure 34 shows the steps taken by the approaches that use behavioral relations.

There are three main differences in the general steps with regard to the replay approach. First, the activity "Assign life cycle transitions" is not automated due to the fact that the approaches based on behavioral relations deal with one-to-many relations between activities and events. Because of this, we cannot assign the activity life cycle transitions automatically, as also other life cycle transitions such as

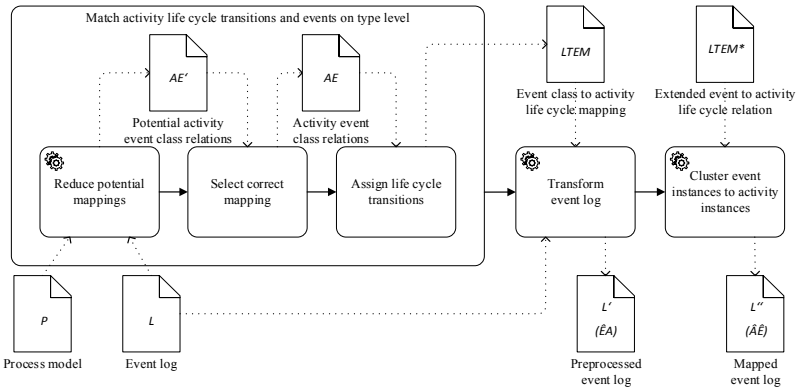


Figure 34: Steps for matching of events and activities using behavioral relations.

start and complete might be involved. These have to be assigned by an analyst in the same way as in the base approach (see Section 3.3.1).

The second main difference to the replay approach is that, as in the base approach, there is a final activity to cluster the event instances to activity instances. Again, this is due to the handling of one-to-many relations between activities and events.

The third and most profound difference is the actual realization of the reduction of potential mappings. We first start, in the same manner as the replay approach, with the definition of variables and their domains. The bijective functions $var : E \to \mathcal{X}$ and $val : A \to 1..|A|$ are defined equally to those from the replay approach. Since the approach based on behavioral relations support one-to-many relations, the example event log L_2 of the order process given in Table 7 is used. The corresponding mapping that contains the one-to-many relations of the event classes from L_2 to the activities of the order process is given in Table 8. The variables for the different event classes are defined

Table 19: Mapping var for the order process with multiple events per activity.

Event $e \in E$	O_CHK_S	O_RC_SA	O_RC_SB	O_RC_E	O_PR_S	O_PR_E	I_SM_E	P_SP_E	P_NOT_E	O_ARC_S	O_ARC_E
Variable $var(e) \in \mathcal{X}$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}

in Table 19. The domains of the variables are the integer–mapped activities of the order process that have been already provided in the previous chapter in Table 18. Having the variables and their respective domains, the solutions to the CSP reflect all possible mappings between events and activities, i.e., for n activities and m events there are potentially n^m solutions. For the used example of the order process these are $7^{11} = 1,977,326,743$ possible mappings. Yet, this also includes solutions where not all activities are assigned to an event or solutions where all events are mapped to one single activity. As

these solutions are not desired, we first restrict the set of solutions to those that assign each activity to at least one event. Note that we assume that the execution of each activity in the process model is being logged by the supporting IT system. Thus, those activities that are not recorded, are not considered in the processing. We assume that each event in the given log relates to exactly one activity in the process model, whereas one activity can relate to multiple events. Thus, we are using the NVALUE constraint, available in many constraint problem solvers (cf. [53]). This constraint ensures that each value in the domain of the variables is assigned at least once. Still, the complexity of the matching problem remains very high. In the following, we present an approach to tackle this complexity issue by combining the information available in the log with knowledge on the process model structure.

To be able to reduce the number of possible mappings, the following sections introduce two different approaches to generate constraints based on behavioral relations. Section 5.3 starts by introducing basic constraints using the notion of behavioral profiles introduced in Section 2.1.5. The subsequent section, Section 5.4, builds on the very same ideas, but uses more expressive Declare rules, which we presented in Section 2.1.6. For better comparison of the two approaches, both make use of the MINERful implementation by Di Ciccio et al. [38] to derive Declare rules from the event log. For the behavioral profile approach, we define a mapping from Declare rules to behavioral relations.

When discovering Declare rules from event logs, these rules can be associated with a reliability metric, namely *support* [78, 36]. Support is a normalized value ranging from 0 to 1 which measures to what extent traces are conforming to a rule. A support of 0 stands for a rule that is always violated. Conversely, a value of 1 indicates that a rule always holds true. In this thesis, we use the definition of support provided by Di Ciccio and Mecella [36]. According to their definition, the analysis of a trace $t_1 = \langle b, a, c, b, a, b, b, c \rangle$, e.g., leads to a support of 1 for *Participation*(a), 0 for *NotCoExistence*(a, b), and 0.75 for *Precedence*(a, b), as 3 b's out of 4 are preceded by an occurrence of a. Considering an event log that consists of t_1 and $t_2 = \langle c, c, a, c, b \rangle$, the support of *Participation*(a) and *NotCoExistence*(a, b) remains equal to 1 and 0, respectively, whereas the support of *Precedence*(a, b) is 0.8 (4 b's out of 5 are preceded by an occurrence of a). For further details of the computation of the support value, we refer to [36].

Having discovered all rules from an event log, all discovered rules having a support lower than a given minimal threshold β are pruned. From our experience, a minimal support of $\beta = 0.9$ has turned out to be the most effective choice. Experimental findings reported in the use cases of [37] confirm this assumption. Yet, the value of β can be redefined by an analyst if needed. Using the same derivation technique

and support calculation for both, the approach based on behavioral profiles and the approach based on Declare rules, eases the comparability.

5.3 DERIVING CONSTRAINTS FROM BEHAVIORAL PROFILES

The previous section explained how the CSP is set up with variables, their domain, and a first constraint ensuring that each value in the domain of the variables is assigned at least once. Coming from this setup, this section shows how to derive further constraints based on behavioral profiles. The work presented in this section is based on our publication [11].

To be able to reduce the number of possible mappings, we look at the behavioral relations between pairs of activities as introduced by behavioral profiles. In a behavioral profile, two activities can be either in strict order, exclusive to each other, or interleaving. Furthermore, the occurrence of one activity may imply the occurrence of another activity, i.e., two activities can be in a co–occurrence relation. Looking at event classes, the same can be said. Weidlich et al. [138] provide an efficient method to calculate behavioral profiles for process models that we used for the implementation of our approach. Yet, for event logs they only provide limited facilities to derive behavioral relations. Therefore, we make use of the MINERful algorithm by Di Ciccio and Mecella [38] to derive declarative rules for pairs of event classes and map these to the mentioned behavioral relations by Weidlich et al. (see Section 2.1.6 for an overview and explanations of the used declarative rules). This procedure also provides better comparability to the approach based on Declare rules that are presented in Section 5.4.

Two event classes k and l are in strict order relation, denoted as $k \rightsquigarrow l$, if there is no trace where an event instance of class k occurs after an event instance of class l. This is expressed by the rule $\mathit{NotSuccession}(l, k)$ derived by the MINERful algorithm. If the two event classes k and l are exclusive to each other, this is represented by the declarative rule $\mathit{NotCoExistence}(k, l)$. Having these two constraints, we can directly conclude the interleaving relation by stating that a pair of event classes (k, l) is in interleaving relation, if neither $\mathit{NotSuccession}(l, k)$ nor $\mathit{NotSuccession}(k, l)$ nor $\mathit{NotCoExistence}(k, l)$ hold with a support higher than the minimum support β. With the Declare rule $\mathit{CoExistence}(k, l)$, we can discover whether two event classes always imply each other. That is, whether they are in a bi-directional co–occurrence relation, i.e., $k \gg l \wedge l \gg k$.

Having the behavioral profiles from both the model and the event log, we can define a number of constraints to reduce the number of possible solutions of the CSP. These constraints are introduced in the following propositional formulas. For each formula, $e_1, e_2 \in E$

denote two different event classes, i.e., $e_1 \neq e_2$. In the same manner, $a_1, a_2 \in A$ denote two different activities, i.e., $a_1 \neq a_2$.

$$e_1 + e_2 \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \implies a_1 + a_2 \quad (2)$$

$$e_1 \rightsquigarrow e_2 \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \implies a_1 \rightsquigarrow a_2 \quad (3)$$

$$e_1 \parallel e_2 \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \implies a_1 \parallel a_2 \quad (4)$$

The general idea behind formula (2), (3), and (4) is that if two event classes are mapped to two different activities, then these two activities have to be in the same type of behavioral order relation. Formula (2) states that when two event classes are exclusive to each other, they can only be mapped to two different activities that are also exclusive to each other. Note that this does not exclude that both event classes are mapped to the same activity. Formula (3) and (4) work analogously for event classes in strict order and for interleaving event classes. The behavioral profile relations ($+, \rightsquigarrow, \parallel$) are exclusive to each other. Hence, only one of the three constraints (2), (3), and (4) is applicable for a particular pair of event classes.

Besides these three basic relations, the causal behavioral profile furthermore contains the co–occurrence relation. For event classes in strict order or interleaving event classes, the constraint expressed in formula (5) may additionally hold.

$$\begin{aligned} e_1 \gg e_2 \wedge e_2 \gg e_1 \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \\ \implies a_1 \gg a_2 \wedge a_2 \gg a_1 \end{aligned} \quad (5)$$

Formula (5) states that if two events that are in a bi–directional co–occurrence relation are mapped to two different activities, these two activities also have to be in a bi-directional co–occurrence relation. Note that we are using a bi-directional co–occurrence constraint. Both events / activities in a bi-directional co-existence relation always imply each other. From now on we will refer to such a bi-directional relation, if we talk about co-existence relations or co-existence constraints.

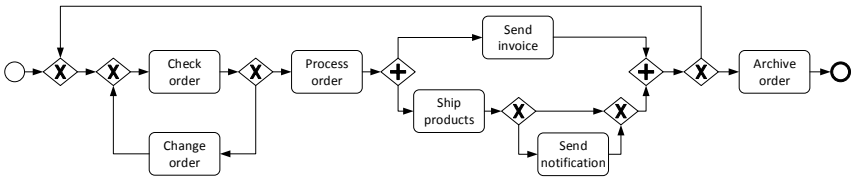


Figure 35: Order process with a loop from end to beginning.

A special case are *larger loops* in a process model. As stated by Armes-Cervantes et al. [3], behavioral profiles face problems with

larger loops. Consider the slightly changed process model of the order process depicted in Figure 35. Directly before the order can be archived there may be a complete repetition of the process. Thus, all activities besides "Archive order" are in a loop and thus, in interleaving order to each other when looking at the behavioral profile. In this case one can make no differentiation between these activities over the behavioral profile and the proposed constraints would not reduce the number of potential mapping significantly. In order to overcome this problem of large loops, we include the direct follower relation, also known as alpha relation or chain succession, where two activities or events directly follow each other in the model or log respectively. We denote a direct follower relation between two events (or activities respectively) as $e_1 \nrightarrow e_2$, if and only if there is a trace $t = \langle \hat{e}_1, \hat{e}_2, \dots \hat{e}_n \rangle$ and $i \in \{1, \dots, n-1\}$ such that $t_i = e_1$ and $t_{i+1} = e_2$ [118, p. 130f]. In order to retrieve the direct follower relations from the event log we employ the two Declare rules ChainPrecedence and ChainSuccession. Formula (6) introduces the corresponding constraint, defining that a pair of event classes that are in direct follower relation can only be mapped to a pair of two different activities if they are also in a direct follower relation.

$$e_1 \nrightarrow e_2 \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \implies a_1 \nrightarrow a_2 \quad (6)$$

5.4 DERIVING CONSTRAINTS FROM DECLARATIVE CONSTRAINTS

The last section presented the approach to build constraints using behavioral profiles derived from a process model and from a corresponding event log. The approach that we introduce in this section builds on the same idea of comparing behavioral relations. One of the major problems when using behavioral profiles for matching are loops, as explained in Section 5.3. We explained how to overcome this problem by using the direct follower relation. Still, they do not fix the information loss regarding short loops of length one. Take for example the short loop between the two activities "Check order" and "Change order" in the order process. In the behavioral profile these two activities are interleaving. Using the direct follower relation does not help either, as both can directly follow each other.

Building in the general idea of the behavioral profile approach, this section introduces an approach that uses more expressive declarative rules as constraints and thereby omits the use of the direct follower relation. Furthermore, the approach is extended to also make use of rules defined over single activities or events respectively. The work presented in this section is based on our publication [10].
In order to reduce the number of possible mappings between activities and events on type level, we look at Declare rules describing the behavior of event logs and process models as defined in Section 2.1.5.

The techniques described in [38] are utilized to derive the described rules from event logs. In order to infer Declare rules from process models, we build upon the following assumption: if an event log is given, such that at least one trace is recorded for each legal path in the process model, then the Declare rules that are discovered out of such a log reflect the behavior of the original process model.[1]

Hence, we can generate an event log from the process model using the simulation technique described in [98], and thereafter apply the discovery algorithm of [38] to derive the Declare rules. We denote the set of all Declare rules inferred from the event log as $\mathcal{B}_L$, and the set of Declare rules discovered from the process model as $\mathcal{B}_P$.

In Section 2.1.6 we already classified the introduced Declare rules into three different categories, namely existence rules $\mathcal{C}_E$, relation rules $\mathcal{C}_R$ and ordering rules $\mathcal{C}_R^{\rightarrow}$. We make use of this categorization by handling all rules that are classified as ordering rules ($\mathcal{C}_R^{\rightarrow}$) as a single rule, giving an ordering between activities / events in a similar way as the strict order relation. Therefore, only the ordering rule with the highest support is kept for each pair of events / activities. Section 2.1.6 elaborated on the fact that particular ordering rules subsume other ordering rules. As a consequence, there may be multiple ordering rules for a pair of activities / events, of which all rules obtain the highest support. One of those rules scoring maximum support is chosen arbitrarily, because we are abstracting from the specifics of the rules and only consider the fact of ordering. In the following, we use $\mathcal{C}_R^{\rightarrow}(e_1, e_2)$ to refer to the chosen ordering relation for a pair of event classes (e_1, e_2) with the highest support. Similarly, $\mathcal{C}_R^{\rightarrow}(a_1, a_2)$ is used for a pair of activities. In contrast to the behavioral profile approach, the use of all available ordering rules in Declare allow to find an ordering relations also for short loops of length one. Due to the use of the `Precedence` and `ChainPrecedence` rules, the relation $\mathcal{C}_R^{\rightarrow}(\text{Check order}, \text{Change order})$ is found for the order process example.

Note that, in contrast to the behavioral profile, declare rules do not capture interleaving semantics explicitly. In declarative languages all activities can be interleaving if not explicitly stated otherwise by a specific rule. Therefore, a set of interleaving events / activities $\mathcal{I} \subseteq (A \times A) \cup (E \times E)$ is introduced. In case there is no ordering rule with a support above β for a given pair of events / activities, we add the pair to the set of interleaving events / activities.

Having the Declare rules from both the model and the event log as well as the set of interleaving pairs of events / activities, constraints to reduce the number of possible solutions of the CSP can be defined. Starting with the ordering rules, formula (7) provides the correspond-

1 Without loss of generality, loops can be unraveled and treated as an optional path that is traversable multiple times.

ing constraint for rules in $\mathcal{C}_R^{\rightarrow}$. If two event classes are in an ordering relation and mapped to two different activities, these activities also have to be in an ordering relation enforcing the same order direction. Note that in formula (7) as well as in all upcoming formulas $e_1, e_2 \in E$ denote two different event classes, i.e., $e_1 \neq e_2$. In the same manner, $a_1, a_2 \in A$ denote two different activities, i.e., $a_1 \neq a_2$.

$$\mathcal{C}_R^{\rightarrow}(e_1, e_2) \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \implies \mathcal{C}_R^{\rightarrow}(a_1, a_2) \tag{7}$$

Following the general pattern of constraints defined for the approach based on behavioral profiles, formula (8) adds the constraint for pairs of event classes that are exclusive to each other and thus, result in a rule of the type NotCoExistence. Again, such a pair of event classes can only be mapped to a pair of exclusive activities or to the same activity.

$$\begin{aligned} \textit{NotCoExistence}(e_1, e_2) \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \\ \implies \textit{NotCoExistence}(a_1, a_2) \end{aligned} \tag{8}$$

Regarding the pairs of events that are not exclusive and for which no ordering rule exceeds the minimum support β, formula (9) ensures that if a pair of interleaving events is mapped to a pair of activities, these activities are also in interleaving order.

$$(e_1, e_2) \in \mathcal{I} \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \implies (a_1, a_2) \in \mathcal{I} \tag{9}$$

The category of relation rules ($\mathcal{C}_R$) further entails the *CoExistence* rule, i.e., the negation of the *NotCoExistence* rule. If two event classes that are co-existing are matched to two different activities, these activities should also be co-existing, as defined in formula (10).

$$\begin{aligned} \textit{CoExistence}(e_1, e_2) \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \\ \implies \textit{CoExistence}(a_1, a_2) \end{aligned} \tag{10}$$

For now, the derived constraints from Declare rules capture almost the same behavior than those derived from behavioral profiles. Yet, there is huge difference when considering loops. Recall the order process example in Figure 35 where a loop was added returning from the end of the process to the begin. In the behavioral profile all pairs of activities within this loop are in interleaving order. In contrast to this, activities that have originally been in strict order before the introduction of the loop are either part of an *AlternatePrecedence*, *AlternateSuccession*, *ChainPrecedence* or *ChainSuccession* rule when taking Declare rules into account. Thus, these pairs still fall into the category of ordering rules. For the approach based on behavioral profiles,

we had to introduce the direct follower relations to regain an ordering relation between those pairs. Yet, the direct following relation is not able to capture distinguishable behavior for short loops of length one, as seen in the order example for activities "Check order" and "Change order". Here, both activities can directly follow each other. The Declare approach is able to make a distinction between the behaviors of the two activities by using the Precedence rule. Furthermore, the direct follower relation also requires activities to directly follow each other and thus, is stricter than the derived declarative rules. This may lead to problems when nonconforming logs come into play.
Besides the already used Declare rules, there are further Declare rules that can be leveraged to build constraints reducing the number of possible solutions. That is, the Declare approach also makes use of the rules classified as existence rules ($\mathcal{C}_E$). The constraint introduced in formula (11) ensures that events for which an *Init* rule exists, are only mapped to activities for which an *Init* rule exists. Formula (12) and (13) work in the same manner for *End* and *Participation* rules.

$$\mathit{Init}(e_1) \wedge \mathrm{Map}(e_1) = a_1 \implies \mathit{Init}(a_1) \tag{11}$$

$$\mathit{End}(e_1) \wedge \mathrm{Map}(e_1) = a_1 \implies \mathit{End}(a_1) \tag{12}$$

$$\mathit{Participation}(e_1) \wedge \mathrm{Map}(e_1) = a_1 \implies \mathit{Participation}(a_1) \tag{13}$$

Having the constraint definitions in the propositional formulas 7-13, a constraint $c_i, i \in 1..|\mathcal{B}_L|$ is added to the CSP for each Declare rule derived from the event log. Coming to an example, consider the rule *Init*(O_CHK_S), which is derived from the event log. In the set of inferred Declare rules from the process model there is only one *Init* rule, namely *Init*(*Check order*). Using the mappings defined in Table 19 and Table 18, the corresponding constraint $c_1 \equiv x_1 = 1$ is derived and inserted into the CSP.

5.5 CONSTRAINTS FOR SPECIAL CASES

Looking at real life event logs, the constraints defined in the previous sections may be too strict in some cases due to the fact that not all behavior of a process is observed equally often. To this end, *interleaving relations*, *co–occurrence relations* can play a special role for both the behavioral profile approach and the Declare approach. Additionally, mandatory events, which are only used in the Declare approach, deserve special attention.

First of all, *interleaving relations* might not always be reflected in the execution. Consider for instance in the order process example of Section 2.3.1. The event classes I_SM and P_NOT_E are in strict order because P_NOT_E always occurs directly before I_SM. Yet, their corresponding activities "Send invoice" and "Send notification" are in interleaving order in the process model. Such a situation is still coherent with respect to the model. Therefore, formula (14) and (15)

introduce a different handling of event classes that are in an ordering relation for the behavioral profile approach and the Declare approach respectively.

$$e_1 \rightsquigarrow e_2 \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \implies (a_1 \rightsquigarrow a_2) \vee (a_1 \parallel a_2) \tag{14}$$

$$\mathcal{C}_R^{\rightarrow}(e_1, e_2) \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \implies \mathcal{C}_R^{\rightarrow}(a_1, a_2) \vee (a_1, a_2) \in \mathcal{I} \tag{15}$$

If two event classes in an order relation are mapped to two different activities, these activities have to be either also in strict order (an ordering relation) or in interleaving order. As the two newly introduced constraints allow for more matchings than their base counterparts formula (3) and formula (7) with respect to the strict order relations in the process model, they are called *relaxed strict order* constraints. We specifically introduce this as a different notion to give the analyst the choice to use the relaxed strict order constraint or the base strict order constraint. The reason for this choice is that the relaxed strict order constraint may introduce quite a number of potential matches that are not wanted because every pair of event classes in strict order that actually maps to a pair of activities in strict order can now also map to all pairs of interleaving activities. If it is known that events belonging to interleaving activities are also seen in all possible orderings equally often, one should not use the relaxed strict order constraint, but rather the constraint defined in formula (3) or formula (7).

In a similar way as for interleaving relations, constraints stemming from *co–occurrence relations* suffer from the fact that some behavior is seen more often than other behavior. Taking again the order example, consider that the event P_NOT_E, which belongs to the optional activity "Send notification", is seen in more than 90 % of the traces. If the minimum threshold β is lower or equal to the relative observations of P_NOT_E, co–occurrence relations with all events that also occur more often in a trace than the threshold are derived. That is, there is for example a co–occurrence relation for P_NOT_E and I_SM, which belongs to the mandatory activity "Send invoice". Yet, in the model, the two activities "Send notification" and "Send invoice" are not in a co–occurrence relation since the former activity is optional. Hence, cases where optional activities are executed almost always, lead to problems with co–occurrence constraints as they disallow the correct mapping. In order to tackle this problem, a relaxed constraint definition for co–occurrence constraints is introduced in formula (16).

$$e_1 \gg e_2 \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \implies !(a_1 + a_2) \tag{16}$$

$$CoExistence(e_1, e_2) \wedge Map(e_1) = a_1 \wedge Map(e_2) = a_2 \implies !NotCoExistence(a_1, a_2) \tag{17}$$

The *relaxed co–occurrence* constraints defined in formula (16) and formula (17) forbid two events that are found to be in a co–occurrence relation to be mapped to two activities that are exclusive to each other. Thereby, the basic co–occurrence constraint is relaxed as we do not require the two matching activities to be in a co–occurrence relation. This allows to handle cases where optional activities are executed overly frequent, while still making use of the co–occurrence relations for the pruning of unwanted mappings.
Looking at *mandatory* events and activities, we face a similar problem as with co–occurrence relation. Consider again the case where the event P_NOT_E, which belongs to the optional activity "Send notification", is seen in more than 90 % of the traces of the order process event log. In this setting a Participation rule is discovered for P_NOT_E and formula (13) leads to the exclusion of the correct mapping. In order to avoid this, formula (13) can be replaced by the following alternative constraint.

$$!Participation(e_1) \wedge Map(e_1) = a_1 \implies !Participation(a_1) \tag{18}$$

Formula (18) states that if there is no Participation rule found for an event, it can only be mapped to an activity for which there is also no Participation rule found in the model. Again, we let the analyst decide whether to use formula (13) or formula (18). In the remainder of this thesis we will refer to formula (18) as *alternative participation* constraint.

5.6 SOLVING THE CONSTRAINT SATISFACTION PROBLEM

Having generated all constraints using either declarative rules or behavioral profiles, the constraint satisfaction problem can already be solved to derive all potential relations between activities and event classes. In contrast to the replay approach, it is not necessary to built an optimization problem in order to deal with noise. Noise is handled already by accepting behavioral relations and declarative rules with a support less than 1.0. Note that this excludes cases where behavioral relations in the event log do not conform to those in the model with a given minimum support. In order to also allow cases where some Declare rules are more often violated than not, there are two ways: *(1) constructing an optimization problem* and *(2) excluding noise-sensitive constraints*. It is also possible to combine (1) and (2). Note

that both are optional steps that should only be taken if necessary, as they both have certain disadvantages. The solution of an optimization problem with all optimal solutions is typically computationally very expensive. The exclusion of certain constraints increases the number of possible solutions, making the computation harder and resulting in a higher number of potential mappings that have to be sorted out. In the following we explain both options.

The *(1) construction of an optimization problem* is done in a similar fashion as for the replay approach. A boolean variable $validCst_i$ is introduced. It is meant to state whether the corresponding constraint c_i is satisfied or not, and is defined as follows:

$$validCst_i = \begin{cases} 1 & c_i = \text{true} \\ 0 & \text{otherwise.} \end{cases}$$

The CSP is then turned into an optimization problem that maximizes the number of fulfilled constraints weighted by their support, $support_i$. Note that there is no support value calculated for interleaving relations, as these are only the result of the absence of an ordering or exclusive relation with at least minimal support. Therefore, there is no weight for interleaving relations, i.e., the support is always 1. Variable $validCstScore$, which is used as optimization goal, is defined as follows:

$$validCstScore = \sum_{i=1}^{|E \times E|} validCst_i \cdot support_i.$$

The optimization problem is then solved retrieving all optimal solutions.

Turning to *(2) the exclusion of certain constraints*, we observed in our validation and evaluation with synthetic and real life event logs, that the interleaving constraints are especially sensitive towards noise. The noise-sensitivity of interleaving constraints is due to the fact that each strict order relations turns into an interleaving relation when it is violated too often to be seen as a strict order relation. Therefore, we make the interleaving constraints optional and let the analyst decide whether to use them or not. The interleaving constraints should only be left out if a log is known to be noisy, as the exclusion of constraints typically increases the number of potential solutions.

5.7 SELECTION OF THE CORRECT TYPE–LEVEL MAPPING

Similar to the replay approach, there can be multiple solutions to the CSP created using behavioral relations. Yet, due to the support of one-to-many relations, there are more patterns that cause a result with multiple solutions. Typically, it is not known how many different

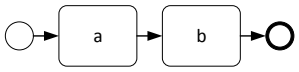


Figure 36: Sequence of activities.

event classes are recorded for each activity, especially in IT systems that are not process-aware. Thus, even a simple sequence of activities as depicted in Figure 36 results in multiple possible solutions. Consider the trace $t_1 = \langle \hat{k}, \hat{l}, \hat{m}, \hat{n} \rangle$ and the simple sequence of activities a and b shown in Figure 36. When matching t_1 and the sequence model, the corresponding CSP returns three solutions. In all three solutions k is matched to a, and n is matched to b. For l and m it cannot be said whether they belong to a or b without further knowledge. It may be that both belong to activity a, or both belong to b, or l belongs to a and m belongs to b. The only mapping that can be excluded, is that l belongs to b and m belongs to a at the same time.

If we want to match t_1 to the model with two concurrent activities shown in Figure 32c, actually every combination of mappings is possible, besides those where all events are mapped to only one of the activities. That is $4^2 - 2 = 14$ different possible mappings for the four event classes and the two concurrent activities. For the matching with two exclusive activities, depicted in Figure 32a, we add the trace $t_2 = \langle p, q, r, s \rangle$. In this case, the CSP returns two solutions: Either k, l, m, n belong to activity a and the rest to b, or the other way around. The approach to deal with multiple solutions of the CSP is the same as the one presented in Figure 33 for the replay approach. The analyst is presented the event class that has the most potentially matching activities. For this event class, the analyst has to choose from the possible matching activities. Afterwards, the defined match is inserted as a new constraint into the CSP, which is consecutively solved again. The procedure is repeated until there is only a single solution for the CSP.

5.8 SUMMARY

In this chapter two approaches for the semiautomatic matching of activities and events on type level have been proposed, namely, the behavioral profile approach and the Declare approach. Similar to the replay approach both approaches build on the base approach and refine the type–level matching by constructing a constraint satisfaction problem. The rationale behind the approaches is that the behavioral relations of event pairs in an event log reflect the behavioral relations of their corresponding activity pairs in the process model. While this is similar to the underlying rationale of the replay approach, the behavioral relations allow for more flexibility in the mapping. That is, also one-to-many relations between activities and events can be covered

both on type and on instance level. With this, Requirement R2 (Different abstraction levels) and Requirement R8 (Hierarchical matching) can be fulfilled by the approaches introduced in this chapter.

In order to build the necessary constraints, the behavioral relations are derived from the event log as well as from the process model. For the constraint definition the approaches rely on the assumption that there is either a one-to-one or a one-to-many relation between activities and events on type level. Hence, many-to-many relations are not supported and Requirement R4 (Shared functionalities) cannot be fulfilled.

Due to the fact that event logs may not always contain all possible execution sequences and dominant orderings or overly frequent executions of optional activities may occur, the event log may entail different behavioral relations than the process model for some activities. These behavioral relations of the event log are not in conflict to those in the process model and therefore require special attention. We therefore introduce relaxed and alternative constraints to account for these special properties of event logs.

Nonconforming behavior (Requirement R9) in the event log is handled by one of two means. First, behavioral relations are discovered from the event log using a support metric for which a minimum threshold is defined. This support metric reflects how often a derived behavioral relation is violated. By setting the threshold below a value of 1.0, also relations that are sometimes violated are discovered. The second means for handling nonconforming behavior is the transformation of the constraint satisfaction problem into an optimization problem. The optimization goal is to maximize the number of conforming behavioral relations in the mapped event log.

6 APPROACH BASED ON LABEL ANALYSIS

This chapter leverages label analysis to provide another semiautomated approach for the type–level matching. The work presented in this chapter has been published in [6] and [8]. The label analysis approach is grounded on the base approach and requires additional assumptions on the input data. These assumptions are described in Section 6.1. Section 6.2 provides an overview of the label analysis approach and the subsequent section details those steps and concepts that are different from the base approach.

6.1 REQUIREMENTS AND ASSUMPTIONS

In line with the assumptions for the base approach, the label analysis approach requires an event log L in the form described in Section 2.1.8 as well as a set of activities A. Again, we assume that the set of activities is contained in a process model as defined in Definition 1, but we do not assume any specified order relation between activities. In contrast to the approaches presented before, the label analysis approach demands all events and activities to carry natural text labels. In addition to an event log and a set of activities, the label analysis approach is able to leverage additional activity descriptions, if available. These activity descriptions may stem, for example, from work instructions, which can be frequently found in industry. While such descriptions improve the matching algorithm, they are not mandatory.

Hence, the label analysis approach has mainly the same assumptions as the base approach, but adds the need for natural language labels for both events and activities. Therefore, the label analysis approach is also designed to fulfill all requirements listed in Section 2.4.

6.2 OVERVIEW OF THE APPROACH BASED ON LABEL ANALYSIS

Like the replay approach and the approach based on behavioral relations, the label analysis approach builds on the base approach and provides a semiautomatic means to match activities and events on type level. Figure 37 shows the steps that are taken by the label analysis approach. In the first step the process model activities are annotated with more details from textual descriptions, if available. The second step computes potential relations between events and activities on type level using the provided annotations. These automatically derived relations have to be refined by a domain expert as input for the next step. Having derived the activity to event class mapping AE,

the procedure continues with the exact same steps defined for the base approach in Section 3.3. The relations to activity life cycle transitions are specified by the analyst and context–dependent mappings are defined where necessary. Using the context–dependent mappings, the log is transformed to reflect the activity to event class mapping. Finally, the event instances, which are already mapped to activities, are clustered to activity instances. The next sections elaborate on the two

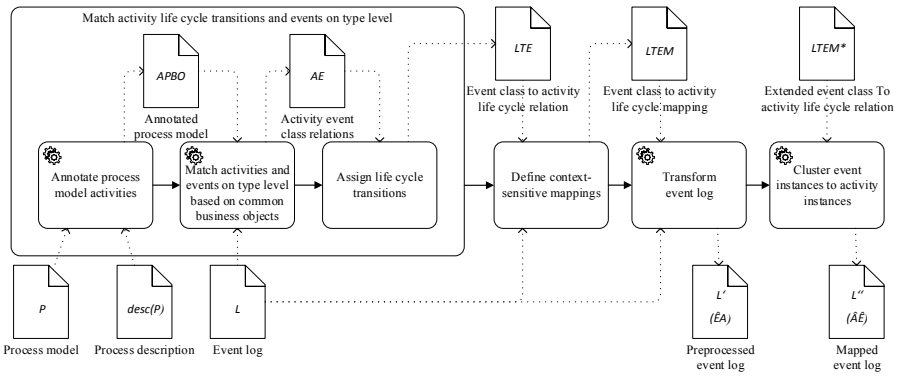


Figure 37: Steps for matching events and activities using label analysis.

newly introduced steps for the annotation of the process model and the consecutive matching of activities and events on type level using common business objects derived from the labels and annotations.

6.3 ANNOTATION OF PROCESS MODEL ACTIVITIES

A major challenge in the matching of activities and events is the diverging level of abstraction, as stated in Requirement R2. In order to bridge the different abstraction layers, we utilize annotations. These annotations serve the purpose of enriching the coarse–granular activities of the process model with detailed information that helps to link to events. In modern business process modeling tools, activities can be connected with more detailed textual descriptions, such that the annotation of the activities is readily available. Often, instructions can also be found in tabular form consisting of columns for the activity name and the detailed description, as in our incident process example in Table 9. In the following, we assume that such a description is available or can be directly linked to an activity.

In order to effectively use the activity descriptions for the matching of event classes and activity types, we have to pre–process the descriptions. As events represent some kind of change to an object, we are especially interested in the objects contained in the activity descriptions. Therefore, the Stanford Part-of-Speech (POS) tagger [63, 112] is used to filter out these objects. The POS tagger parses natural text and assigns each word to its part of speech, e.g., verb, noun, article, adjective, etc. From these categories we only take into account words

that are nouns or words for which no real category can be found by the POS tagger. The latter are most often abbreviations as, e.g., "CI" or foreign words. Furthermore, all numbers are filtered out. The goal is to extract potential business objects. The set of all potential business objects is denoted as PBO. $PBO_a \subset PBO$ is the set of potential business objects $pbo_i \in PBO_a$ that unites all potential business objects for an activity $a \in A$. These objects are extracted from all activity description $ad \in desc(a)$. Additionally, the labels of the activities are processed in the same way to extract further potential business objects. The activities are annotated with the derived objects for further processing in the next phase of the approach. The result of this phase is an activity annotation relation $APBO \subseteq A \times PBO$.

Table 20: Potential business objects derived for the incident process example.

Activity	Potential business objects
Incident logging	logging, member, level, details, group, incident
Incident classification	details, classification
Initial diagnosis	diagnosis, knowledge, level, supporter, configuration, item, base, problem, CI
Functional escalation	escalation, level, knowledge, supporter, ticket, route, base, team, security, group, solution, incident
Investigation and diagnosis	level, supporter, configuration, item, diagnosis, group, investigation, solution, incident, CI
Security incident handling	security, handling, incident, system, level, issue, IT, attack, security, group, solution, actions, database
Resolution and recovery	resolution, recovery, knowledge, base, customer, solution, incident
Incident closure	closure, configuration, item, documentation, incident, CI

This relation is a many-to-many relation, i.e., one activity can be linked to multiple potential business objects, and one potential business object can be associated with multiple different activities. Note that the annotation is not mandatory for each activity. Yet, it presumably improves the automated matching result because the textual descriptions are likely to be closer to the abstraction level of the event log than the activities in the process model as we show in our evaluation in Section 8.4.

6.4 MATCHING ACTIVITIES AND EVENTS BASED ON COMMON BUSINESS OBJECTS

Having annotated the activities with their potential business objects, the next step deals with the derivation of the activity to event classes relation AE. To this end, we inspect each combination of event class and activity name as well as each combination of event class and activity description for potential correspondences.

Algorithmus 3 : Derive potential event–activity relation.

```
1: checkRelation(EventLog L, ProcessModel P, ProcessDescription
   desc(P))
2: Set AE := ∅
3: Set APBO := ∅
4: Set EPBO := ∅
   {Annotate activities with potential business objects}
5: for all a ∈ A do
6:    Set PBO_a := PBO_a ∪ extractNouns(a)
7:    for all ad ∈ desc(a) do
8:       Set PBO_a := extractNouns(ad)
9:    end for
10:   APBO := APBO ∪ {(a, pbo) | pbo ∈ PBO_a}
11: end for
   {Annotate event classes with potential business objects}
12: for all e ∈ E do
13:   Set PBO_e := extractNouns(e)
14:   EPBO := EPBO ∪ {(e, pbo) | pbo ∈ PBO_e}
15: end for
   {Check matches on business object level}
16: for all (e, pbo_e) ∈ EPBO do
17:   for all (a, pbo_a) ∈ APBO do
18:      if checkBusinessObjectMatch(pbo_a, pbo_e) then
19:         AE := AE ∪ {(e, a)}
20:      end if
21:   end for
22: end for
23: return AE
```

Algorithm 3 details the general procedure. The algorithm takes an event log, a process model and a process description. First, we derive the sets of potential business objects for activities and events. The function *extractNouns* uses the Stanford POS tagger to return all potential business objects as explained above. In order to check for potential correspondences, we also derive the objects from the event classes in the same manner, yielding the relation $EPBO \subseteq E \times PBO$ (line 12–15). Each tuple in APBO is compared to each tuple in EPBO by comparing the business objects (line 16-22). As we aim for a high recall, we do not only make simple string comparisons in order to check the relatedness of two business objects. Rather, we employ natural language processing techniques as we explain in the following.

Since we evaluate our approach in a context using the German language, we need to pay special attention to this language. Nonetheless, the basic techniques are also available for many other languages. Looking at the German language we face two potential challenges: word form variance and compound words. German is a morphological complex language having a high variance in word forms expressed by many cases and inflections (cf. [65]). Looking at nouns,

for example the word "Buch" (book) transforms to "Bücher" in the plural form or to "des Buches" for the genitive case. Regarding compound words, in German these are single words created by concatenating several words to a new word, e.g., "Fach|gruppe" (professional group).

In order to address these two challenges, two techniques from the natural language processing (NLP) area have been proven beneficial: stemming and word decomposition [21]. Stemming refers to the reduction of derived word forms to a common stem, e.g., "Grupp" for "Gruppe" and "Gruppen". In the implementation of our approach we use the stemming functionality of the Apache Lucene project[1]. For the decomposition of compound words, we use a language independent, lexicon-based approach developed by Abels and Hahn [1]. It generates possible splittings of words and checks whether the generated parts are covered in a lexicon. In our approach we use JWordSplitter, an open source implementation of this approach with an integrated German lexicon[2].

Algorithmus 4 : Check for business object matches.

```
1:  checkBusinessObjectMatch(pbo1, pbo2)
2:  if eventObject = textObject then
3:     return true
4:  else
5:     Set wordParts1 := decompose(pbo1)
6:     Set wordParts2 := decompose(pbo2)
7:     for all wp1 ∈ wordParts1 do
8:        for all wp2 ∈ wordParts2 do
9:           if stem(wp1) = stem(wp2) then
10:             return true
11:          end if
12:       end for
13:    end for
14: end if
15: return  false
```

Algorithm 4 illustrates the procedure to check for a relation between two business objects. First, we conduct a simple string match (line 2) and second, we decompose the business objects into their smallest semantic components and compare these with one another (line 5-14). The comparison of decomposed word parts is done by comparing the word stems. In this way we are able to relate words such as "Fachgruppe" (professional group) and "Skillgruppen" (skill groups).

The result of the described steps is an automatically provided list of potential activity to event class relations on type level (AE) that can be refined by a domain expert. Refining in this case means that the ex-

1 See http://lucene.apache.org.

2 See http://www.danielnaber.de/jwordsplitter/.

pert has to identify and remove incorrect relations and add relations that are missing.

Table 21: Potential type–level relation of activities and event classes of the incident example.

Activity	Event classes
Incident logging	*Group*, *Details*
Incident classification	Details, *Classification*
Initial diagnosis	*CI*
Functional escalation	*Group*, Solution
Investigation and diagnosis	*Group*, Solution, *CI*
Security incident handling	*Group*, Solution
Resolution and recovery	*Solution*
Incident closure	*CI*

The potential type–level relation of activities and event classes for the incident process example is shown in Table 21. The correct matching event classes are highlighted in italics. All other found relations need to be identified and removed by the analyst. Furthermore, it can be seen that not all relations were found. The relation of the event class "Status" to the closure of the incident and the relation from the event class "CI" to the activity "Security incident handling" need to be added manually.

6.5 SUMMARY

This chapter introduced the third approach to integrate a semiautomatic means to match events and activities on type level into the base approach. In contrast to the previously proposed approaches, the label analysis approach does not depend on behavior but leverages semantics from the labels of events and activities for the matching. In order to bridge the gap between the mostly higher abstraction level of the process model towards the lower abstraction level of the event log, the process model activities are annotated with descriptions from work instructions which are typically on an intermediate abstraction level. The label analysis approach leverages natural language processing techniques in order to extract the business objects from event and activity labels as well as from the annotated descriptions. Furthermore, stemming and word decomposition are applied to overcome matching challenges imposed by morphological complexity of a language. The focus here lies on the German language. Nonetheless, the employed techniques are also available for most other languages.

Since the label analysis approach does not rely on behavior, it is resilient to noise by its design and therefore fully supports Requirement R9 (Nonconforming execution). Furthermore, there are no constraints towards the cardinality of the relation between activities

and events. Thus, the label analysis approach also supports Requirement R2 (Different abstraction levels) and Requirement R4 (Shared functionalities), which have not been supported by the previously introduced type–level matching approaches.

7 INTEGRATED APPROACH

So far, different matching approaches based on behavior and an approach based on label analysis have been introduced to provide a higher level of automation for the matching of activities and events. In this chapter, we propose means to integrate different type–level matching approaches. Grounded on the base approach, we elaborate on additional assumptions in Section 7.1, where we furthermore clarify the fulfilled requirements. In Section 7.2, a general concept of combining different approaches is provided. Based on this, Section 7.3 uses the integration of the Declare approach (see Chapter 5) and the label analysis approach (see Chapter 6) as an example that forms an integrated approach of behavioral analysis and label analysis. The idea of the integrated approach is to facilitate integration for any type of matching facility that allows for the matching of events and activities on type level. The goal of this integration is to further reduce manual work for the matching of activities and events on type level.

7.1 REQUIREMENTS AND ASSUMPTIONS

The integrated approach combines two approaches for the matching of event classes and activities under the general line of the base approach. As a consequence, the assumptions made for the integrated approach unite the assumptions made for these combined approaches. Wherever one of the approaches imposes stronger assumptions on the inputs, these assumptions have to be taken for the integrated approach. In line with this, the weakest approach in use also dictates the requirements that can be fulfilled.

Thus, looking at the exemplary combination of the Declare and the label analysis approach, the integrated approach requires an event log and a Petri net for which all events and activities use natural language labels. Furthermore, the Petri net needs to be bounded and an initial and at least one final marking have to be defined. All transitions that represent activities have to be part of at least one sound firing sequence. While the event log must contain at least one event for each activity in the process model, it must not contain event classes that belong to no or to multiple activities. That is, the integrated approach for the combination of Declare and label analysis approach does not support missing events (Requirement R5), additional events (Requirement R6) and shared functionalities (Requirement R4). Nonetheless, one-to-one and one-to-many relations are handled (Requirement R7, Requirement R1, and Requirement R2). As the Declare approach is used in the exemplary integrated approach, we have to assume that

there are no additional events, which cannot be mapped to any activity. That is, Requirement R6 cannot be handled directly by the integrated approach. For the use of the Declare approach, the execution recorded in the given event log needs to conform to the provided process model such that all Declare rules present in the model can be rediscovered from the event log with a support higher than a given minimal threshold.

Regarding requirements that are not directly bound to the input data, the integrated approach delivers the same fulfillment level as the base approach. For example hierarchical mappings can be obtained with multiple iterations of the approach. Requirement R8 can, hence, be fulfilled using the integrated approach.

7.2 GENERALIZED INTEGRATION OF MULTIPLE TYPE–LEVEL MATCHING APPROACHES

The integrated approach aims at combining different approaches for the matching of events and activities on type level. Similar to the specialized type–level matching approaches that have been introduced in this thesis, the integrated approach grounds on the base approach and provides automation for the type–level matching.

Figure 38 depicts the general procedure of the integrated approach. The integrated approach uses the very same four steps as the base approach. Of these four steps, only the implementation of the matching of activity life cycle transitions and events on type level differs from the base approach. First of all, the type–level matching takes additional data as input, besides the event log and the process model. Such additional data can for example contain process descriptions, which are used by the label analysis approach.

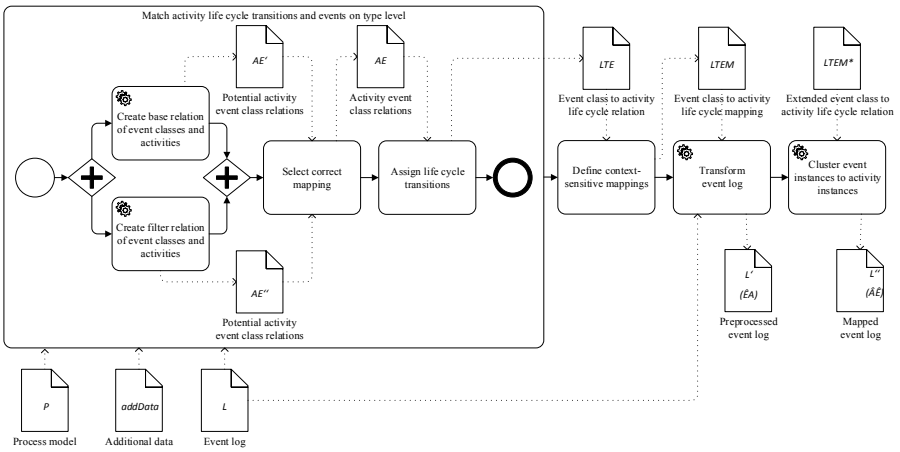


Figure 38: Integrated approach for the matching of events and activities.

The main procedural difference to the previously introduced type-matching approaches is that we are concurrently generating two sets

of potential activity to event class relations ($Æ'$ and $Æ''$). The generation of two sets originates from the insight that different approaches for the type–level matching vary in terms of coverage with respect to a final mapping. That is, for some approaches the set of potential activity to event class relations may not include all relations required for the final mapping. Looking at the type–level matching approaches introduced in this thesis, it can be seen that the behavioral matching approaches are designed to always include the complete final relations of activities and event classes in their potential activity to event class relation. This is due the fact that the approach based on behavior start from all possible relations and prune these relations by eliminating impossible combinations. If the assumptions made by these approaches are fulfilled, the correct relation is always included in the set of potential relations. For the label analysis approach, this cannot be said. The label analysis approach starts with an empty set and adds those relations that can be found over the matching of extracted business objects. It may happen that not all relations of event classes and activities can be found, as we also show in our evaluation in Section 8.4.

Based on this insight, we part type–level matching approaches into those that are able to always include the correct and complete type–level relation, and those that cannot. Note that "always" here refers to "always under the given assumptions". The first set of type–level matching approaches can be used to produce the *base relation* ($Æ'$) of activities and event classes while the latter can deliver a *filter relation* ($Æ''$).

As depicted in Figure 38, both potential activity to event class relations serve as input for the selection of the correct mapping. We extend the questionnaire-driven selection approach used for the approaches based on behavioral relations, by using the relation AE'' as a filter when presenting the activities between which the analyst has to choose. In case an event class e is mapped to multiple activities over all relations contained in the base relation AE', the analyst has to inspect which of these multiple activities are correct mappings. Having both relation AE' and AE'', the analyst will only be presented the activities that have a mapping to event class e in both relations. We denote the set of activities potentially mapped to event class e in the base relation as $A'_e = \{a'_e \mid \exists (a'_e, e) \in AE'\}$. Similarly, the derived activities for e contained in the filter relation are denoted as $A''_e = \{a''_e \mid \exists (a''_e, e) \in AE''\}$. The set of presented activities for event class e is defined as $A^*_e = A'_e \cap A''_e$. Depending on whether all of the integrated type–level matching approaches support shared functionalities, the analyst can select one or more matching activities from the presented set.

Due to the fact that the relation AE'' may not contain the correct mapping, it can happen that also A_e^* does not contain the correct matching activities for event class e. Therefore, the analyst can indicate that there are missing matches. Consequently, a new set of activities is presented from which set the analyst can complete their choice. This second set of activities is defined as $A_e^{**} = A_e' \setminus A_e''$ and contains only those activities found in a relation to event class e in A_e'. As it holds that $A_e' = A_e^* \cup A_e^{**}$, the correct activities have to be contained in the two presented sets. By splitting the set of activities that an analyst has to inspect, the selection step is made easier as less information has to be processed at the same time.

Having derived the activity event class relations $Æ$, the analyst needs to assign activity life cycle transitions to the relations in $Æ$. From here the procedure continues in the same manner as for the base approach. The analyst needs to define context–sensitive mappings for those event classes that stem from shared functionalities. This completes the event class to activity life cycle mapping, with which the event log is subsequently transformed. Finally, the analyst has to provide the required instance border conditions for the clustering of activity instances.

7.3 INTEGRATING THE DECLARE AND THE LABEL ANALYSIS APPROACH FOR THE TYPE–LEVEL MATCHING

This section exemplarily describes the integrated approach for the combination of the Declare and the label analysis approach. As explained in the previous section, the integrated approach also builds on the base approach and provides a semiautomatic means to match activities and events on type level. When combining the Declare and the label analysis approach, we have to take all assumptions of both approaches into account, as discussed in Section 7.1. For overlapping assumptions that refer to the same aspect, the stronger assumption needs to be considered. Therefore, the integration of these two approaches cannot handle shared functionalities and the step "Define context–sensitive mappings" becomes superfluous and is removed in the same way as for the Declare approach itself.

Figure 39 depicts the detailed steps of the type–level matching of activities and events on type level. The processing in the integrated approach starts with a parallel split where the upper branch corresponds to the reduction of potential mappings as described in Section 5.4. A constraint satisfaction problem based on Declare rules derived from process model and event log is built and solved to achieve this reduction. The lower branch corresponds to the matching of event classes and activities based on common business objects derived using the label analysis techniques provided in Chapter 6. The result of

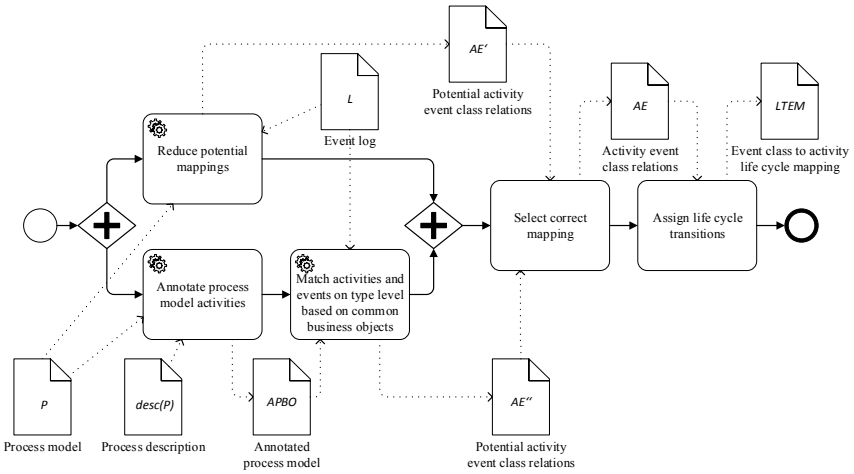


Figure 39: Integrated approach to match activity life cycle transitions and events on type level.

the two branches are the two different activity event relations, AE' and AE''.

From the design of the label analysis approach it is known that the provided set of potential activity to event class relations does not necessarily include all pairs of activities and event classes required to build the final relation AE. This will also be later seen in the evaluation, where the calculated recall of discovered type–level relations is lower than one. Hence, a filtering of AE'' will never lead to a complete mapping. In contrast, the relation AE' derived by the Declare approach always entails the correct mapping, if all assumptions are met. Thus, AE' is used as the base relation for the selection while AE'' is set as the filtering relation. Due to the fact that we cannot consider shared functionalities, the analyst has to select exactly one activity for each event class presented in the subsequent selection phase.

Having the final relation AE the integrated approach continues in the same way as the described in the previous section. The analyst has to provide the mapping to specific life cycle transitions, resulting in the relation LTE. As we do not allow shared functionalities, there is no need to create context–sensitive mappings and the event log can be directly transformed using the event class to activity life cycle mapping $LTEM$. Finally, the user needs to provide activity instance border definitions that are then used to do the clustering of activity instances.

To provide an example for the integration of the Declare and the label analysis approach, we provide a few modifications to our incident process example. Consider the slightly modified event log of the incident process in Table 23. The event log has been adapted in such a way that the Declare approach can be applied. To achieve this, the events of the classes “Group” and “CI” have been changed by adding

further context information to their name attribute, thereby creating new event classes, such as "Group – 1st level first" or "Group – 2nd level". With this change, shared functionalities, which cannot be handled by the Declare approach, were removed. In total, the new event log comprises twelve event classes. Moreover, some changes in the ordering of the events have been made in order to fulfill the minimum conformance assumption made by the Declare approach.

With the process model from Figure 14, the activity descriptions from Table 9, and the event log from Table 23, the integration of the Declare approach and the label analysis approach can be demonstrated. The type–level matching of the Declare approach returns 40 different solutions to the created CSP, i.e., 40 different potential mappings. Table 22 shows the event classes and their derived potential activities in the order in which the Declare approach determines the questioning. In total, five questions need to be asked to arrive at the final mapping. The second column of Table 22 entails the potential activities derived by the Declare approach. That is, column one and two capture the base relation AE'. The filtering relation AE'' derived by the label analysis approach is very similar to the relation shown in Table 21 in the previous chapter. Only the event classes "Group" and "CI" need to be replaced by their new modified versions. Using this filtering relation, the third column of Table 21 is derived. For the event classes "Group – 1st level first" and "CI – 1st level solved" the use of the filtering relation results in a reduction of potential activities that are shown to the analyst. Instead of two activities for each event class, the analyst only has to check whether the one shown activity is the correct one. In both cases, the correct activity is offered so that the analyst can quickly decide. Having answered the questions for the five event classes, the correct mapping is derived.

Table 22: Event classes in question with potential activities derived by the Declare approach and reduced potential activities shown in the integrated approach with label analysis.

Event class	**Potential activities (Declare)**	**Reduced potential activities**
Group – 1st level first	Incident classification, Incident logging	Incident logging
Group – Security	Investigation and diagnosis, Security incident handling	Investigation and diagnosis, Security incident handling
Group – 1st level	Investigation and diagnosis, Functional escalation	Investigation and diagnosis, Functional escalation
CI – 1st level solved	Resolution and recovery, Incident closure	Incident closure
Details	Incident classification, Incident logging	Incident classification, Incident logging

Table 23: Conforming example log for the incident process without shared functionalities.

Variant	Event class	Value	Role	Activity
t_1	Group - 1st level first	1st-SAP	1st level	Incident logging
t_1	Details	SAP is offline	1st level	Incident logging
t_1	Details	SAP R3 is offline	1st level	Incident logging
t_1	Classification	SAP	1st level	Incident classification
t_1	Solution	Cleared cache	1st level	Resolution and recovery
t_1	Status	Closed	1st level	Incident closure
t_2	Group - 1st level first	1st-Intra	1st level	Incident logging
t_2	Details	Password forgotten	1st level	Incident logging
t_2	Classification	Password	1st level	Incident classification
t_2	CI - 1st level	US1234	1st level	Initial diagnosis
t_2	Solution	Password reset	1st level	Resolution and recovery
t_2	Status	Closed	1st level	Incident closure
t_3	Group - 1st level first	1st-Mail	1st level	Incident logging
t_3	Details	Cannot send mails	1st level	Incident logging
t_3	Classification	Mail	1st level	Incident classification
t_3	CI - 1st level	Outlook Client	1st level	Initial diagnosis
t_3	CI - 1st level	Outlook Server 1	1st level	Initial diagnosis
t_3	Group - 1st level	2nd-Mail	1st level	Functional escalation
t_3	CI - 2nd level	Outlook Server 2	2nd level	Investigation and diagnosis
t_3	Group - 2nd level	1st-Mail	2nd level	Investigation and diagnosis
t_3	Solution	Change client settings	1st level	Resolution and recovery
t_3	CI - 1st level solved	Outlook Client	1st level	Incident closure
t_3	Status	Closed	1st level	Incident closure
t_4	Group - 1st level first	1st-generic	1st level	Incident logging
t_4	Details	Virus found	1st level	Incident logging
t_4	Classification	Virus	1st level	Incident classification
t_4	CI - 1st level	Notebook 325	1st level	Initial diagnosis
t_4	Group - 1st level	Security	1st level	Functional escalation
t_4	CI - Security	Station 133	Security	Security incident handling
t_4	CI - Security	Win-LP231	Security	Security incident handling
t_4	Group - Security	1st-generic	Security	Security incident handling
t_4	Solution	Virus removal	1st level	Resolution and recovery
t_4	CI - 1st level solved	Win-LP232	1st level	Incident closure
t_4	Status	Closed	1st level	Incident closure
t_5	Group - 1st level first	1st-monitor	1st level	Incident logging
t_5	Details	Disk space shortage	1st level	Incident logging
t_5	Classification	Backup	1st level	Incident classification
t_5	CI - 1st level	Backup Server 1	1st level	Initial diagnosis
t_5	CI - 1st level	HD 142	1st level	Initial diagnosis
t_5	Solution	Changed broken HD	1st level	Resolution and recovery
t_5	CI - 1st level solved	HD 157	1st level	Incident closure
t_5	Status	Closed	1st level	Incident closure
t_6	Details	High network traffic	1st level	Incident logging
t_6	Group - 1st level first	1st-monitor	1st level	Incident logging
t_6	Solution	Short peak. No action	1st level	Resolution and recovery
t_6	Status	Closed	1st level	Incident closure

7.4 SUMMARY

Within this chapter, we proposed a means for the integration of different type–level matching approaches. Once more, the introduced approach is grounded on the base approach and details the concrete implementation of the type–level matching. We put forward a generic way of integrating two different type–level matching approaches and exemplify the generic approach by the integration of the Declare approach and the label analysis approach. The integration relies on the characteristic of matching completeness and divides the type–level approaches into two categories: approaches that always contain the complete correct mapping, and those that may only contain parts of the mapping. By "always" we refer to "always under the stated assumptions of the approach".

The integrated approach adopts the questioning phase known from the behavioral approaches and leverages the resulting type–level relations from two different approaches for this phase. An approach that is designed to always deliver the complete mapping — such as the behavioral approaches — can be to construct a base type–level relation of activities and events. Approaches of the second category can be used to derive a filtering relation that overlays the base relation in order to present only those relations covered by both approaches. Thereby it is possible to present fewer potential activities to the analyst when it is necessary to make a manual decision for a certain event class. Nonetheless, the integrated approach does not restrict the resulting type–level relation to the intersection of the two integrated approaches. It still allows to view all found event–activity relations from the base relation in case the intersection does not contain all correct mappings. With this, the integrated approach remains flexible while still easing the type–level matching once more.

Part III

EVALUATION AND CONCLUSION

8 EVALUATION

Having introduced the different approaches for the mapping of activities and events, this chapter focuses on the implementation, validation and evaluation of the defined concepts. Section 8.1 starts by giving an overview of the implemented ProM plug–ins. These plug–ins are then used in the subsequent sections to validate and evaluate the different approaches. In Section 8.2, we report on the results of two case studies in which we evaluated the base approach with real life data. A validation of the behavioral approaches is performed with synthetically created event logs from a large set of industry process models in Section 8.3. Sections 8.4 and 8.5 provide the evaluation of the label analysis approach and the integrated approach with case studies. For the evaluation of the integrated approach the Declare and the behavioral profile approach have been combined with the label analysis approach. Section 8.6 concludes this chapter with a comparison of the introduced approaches.

8.1 IMPLEMENTATION

For the purpose of evaluation, we implemented the introduced approach for the matching of events and activities in the ProM framework[1]. ProM is an extensible open-source framework for process mining [130]. It has become very popular in the research field of process mining for the implementation and evaluation of various process mining techniques. The implementation of our work is using version six of the ProM framework. Version six has been introduced in [133] and provides a plug–in architecture where functionality and graphical user interface are clearly separated. Plug–ins can be grouped into packages, which can be easily installed using a package manager. Version six of ProM has been the first process mining tool that uses the eXtensible Event Stream (XES) format for event logs. The XML-based format XES addresses the large variety of available log information and introduces a standard for storing and exchanging event logs [133]. XES is also supported by other popular process mining tools like Disco[2]. In version six of ProM, the concept of an object pool has been introduced [133]. Plug–ins can use objects from this pool as input and create new objects that are given back into this pool.

All plug–ins that have been developed for the evaluation of the concepts introduced in this thesis can be found in the publicly avail-

1 See http://processmining.org/prom/start.
2 See http://fluxicon.com/disco.

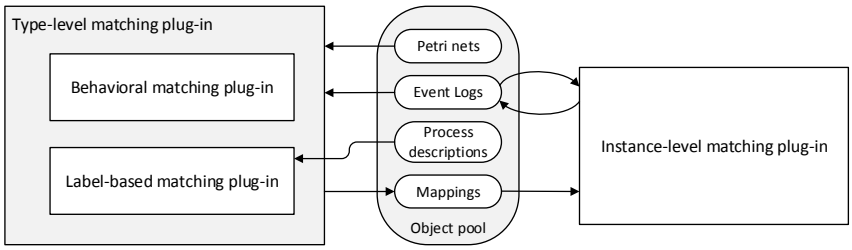


Figure 40: FMC Block diagram of the implemented ProM plug–ins with inputs and outputs.

able ProM package "Event2ActivityMatcher" [3]. Figure 40 depicts a FMC Block diagram [4] that gives an overview of the ProM plug–ins implemented for this thesis. The mandatory inputs for the type–level mapping plug–in are an event log and a Petri net. Optionally, a process description, which can be used by the label analysis approach, may be provided. In ProM this results in two different plug–in variants as shown in Figure 41. On the left side the required inputs are listed. The first variant uses all three inputs while the second one requires only an event log and a Petri net.

Figure 41: ProM action screen showing the type–level mapping plug–in.

The type–level plug–in provides the choice to use one of the behavioral approaches, the label analysis approach, or the integrated approach, which is a combination of both. Both the label analysis and the behavioral approaches are implemented as separate plug–ins to make them independently usable. The type–level plug–in provides a configuration screen to choose between the different mapping ap-

3 The source code is available in the subversion repository at https://svn.win.tue.nl/repos/prom/Packages/Event2ActivityMatcher.

4 Fundamental Modeling Concepts (FMC) is a modeling notation where Block diagrams are used to illustrate compositional structures as a composition of collaborating system components. For an introduction into FMC see [68].

proaches and provides the capabilities for their integration (see Figure 42).

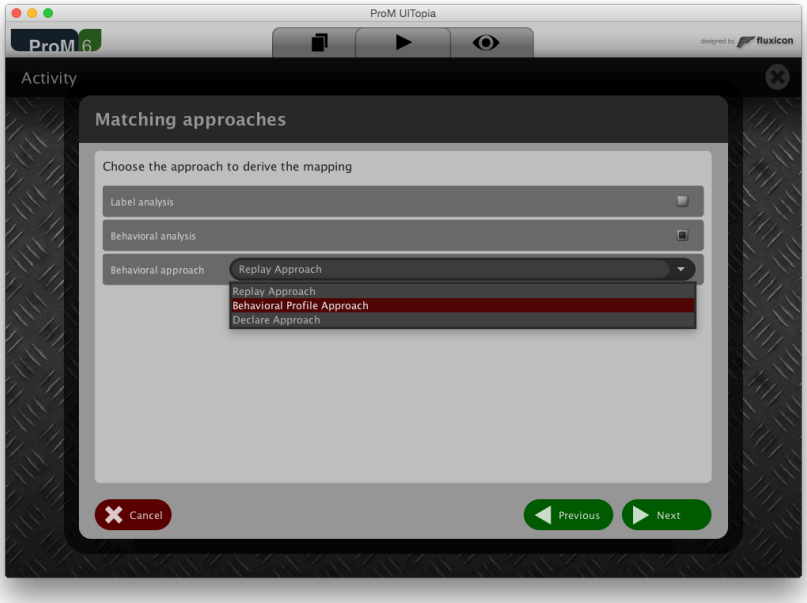


Figure 42: Configuration screen for the type–level mapping plug–in.

For each of the approaches, another configuration screen allows the user to choose between the different settings that have been explained in this thesis. Figure 43 exemplarily depicts the configuration screen for the Declare approach. In this configuration screen one can choose whether to use relaxed strict order constraints, alternative participation constraints, and interleaving relations. Furthermore, it is possible to decide if an optimization problem shall be constructed or not. While there are many implementations of constraint satisfaction problem solvers, the Java library CHOCO[5] [64] has been used for all CSP implementations required by the behavioral matching plug–in. CHOCO has been chosen because it can be easily integrated into Java applications and because it proved to be among the fastest CSP solvers in recent competitions like MiniZinc[6].

The specific matching plug–ins return the potential event–activity relations, which are used by the type–level mapping plug–in to guide the user to the correct event–activity mapping. This guidance is provided using the explained questionnaire-driven approach. Figure 44 shows the screen where the analyst is asked to choose the correct activity for an event class.

Once the potential mappings are reduced to one, the type–level mapping plug–in produces an object of the type "EventsToActivi-

5 See http://www.emn.fr/z-info/choco-solver/.
6 See http://www.minizinc.org/challenge2014/results2014.html and http://www.minizinc.org/challenge2013/results2013.html.

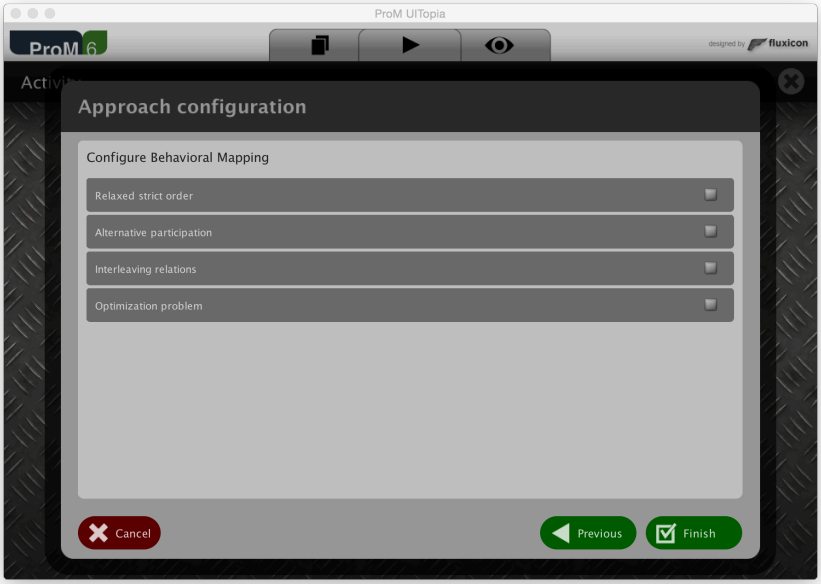


Figure 43: Configuration screen for the behavioral profile approach.

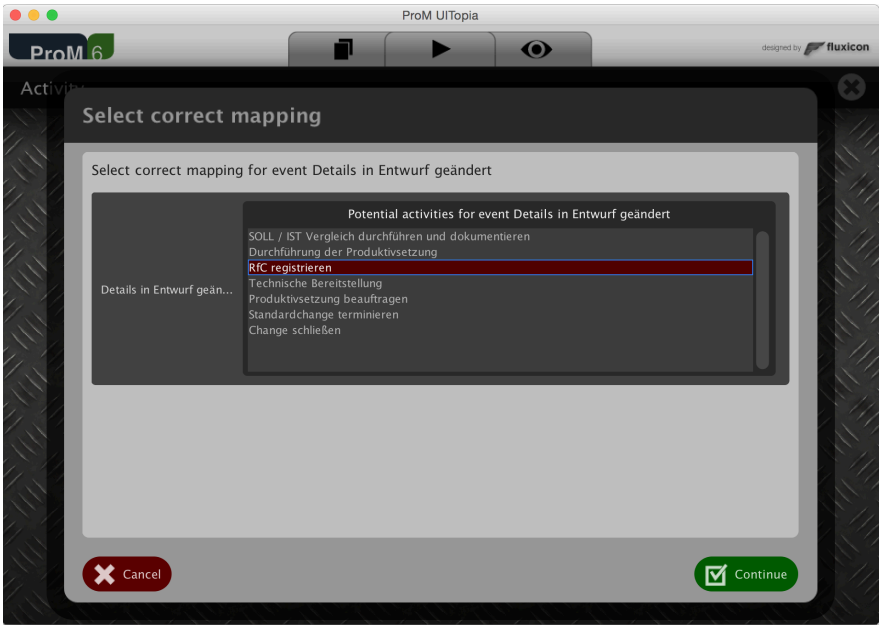


Figure 44: Choosing the correct activity for an event in the ProM plug–in.

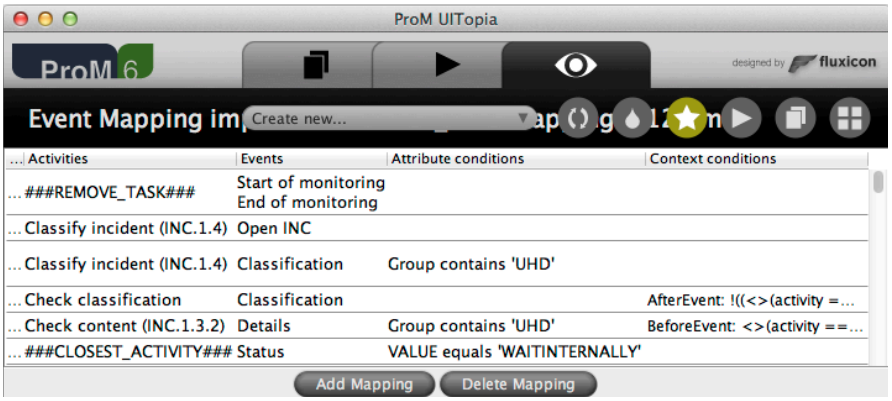


Figure 45: Screenshot of mapping in ProM.

tiesMapping" (short "Mapping") that is returned to the object pool of ProM and can be subsequently used by the instance-level mapping plug–in, as illustrated in Figure 40. The object of type "EventsToActivitiesMapping" created by the type–level mapping plug–in contains only the activity event relation AE. Thus, the user needs to provide the required mappings to life cycle transitions and context conditions if necessary. This additional information can be added in the view interface of the "EventsToActivitiesMapping" object. Figure 45 shows a screenshot of an excerpt of a mapping in this view. The view allows for adding, editing and removing mapping definitions containing attribute and context conditions. The mappings can be stored into and loaded from XML files. An example of the XML representation is shown in Figure 46. Here, also the instance border conditions can be specified for each mapping.

```
<EventMapping>
    <activity><name>Check content (INC.1.3.2)</name></activity>
    <eventName>Details</eventName>
    <InstanceBorder>
        <className>SourceEventLoopDifferentResource</className>
    </InstanceBorder>
    <metaDataCondition>
        <name>Group</name>
        <value>UHD</value>
        <type>contains</type>
    </metaDataCondition>
    <eventCondition location="BeforeEvent"><![CDATA[
        <>(activity == "Change of Group");
    ]]></eventCondition>
</EventMapping>
```

Figure 46: Excerpt of the mapping definitions in XML.

Once all required conditions have been specified in the object of type "EventsToActivitiesMapping", the instance-level mapping plug–in takes the event log and the mapping definitions and produces a

new event log where all event instances are mapped to their corresponding activity instances.

Besides the graphical ProM user interface for the plug–ins, we furthermore developed a standalone command line application[7]. This application serves the purpose of simulating user interaction and automatic execution of experiments on a cluster. The command line interface gives further options, such as defining a maximum duration for the solving of a CSP and the minimum support threshold β. A complete list of options can be obtained using the "–help" flag of the command line application.

8.2 EVALUATION OF THE BASE APPROACH

Having described the implementation in the previous section, this section describes the evaluation of the base approach.

8.2.1 *Evaluation Goals and Setup*

As stated in Section 3.3, the base approach is designed to address all requirements described in Section 2.4. The goal of the evaluation is to verify that all requirements can be met. To demonstrate this, we conducted two industry case studies at a large German IT outsourcing company. For these case studies the base approach has been applied to two processes: the incident management and the change process.

Breaking down the overall goal of verifying that the outlined requirements can be met in real life case studies, we define the following sub-goals. To this end, we want to assess the feasibility and correctness of the application of our different concepts that relate to the defined requirements. With feasibility we refer to the possibility to actually formulate all required mappings with the introduced concepts of this thesis. The correctness is assessed by domain experts who inspect the actual results and check whether individual events have been mapped correctly. Thus, a gold standard mapping is created by manually checking the results of the base approach.

For the evaluation, all four steps of the base approach are carried out to map the events from the extracted event logs to their corresponding activity life cycle transitions found in the process models. The base approach generates two different event logs: First, an intermediate event log L', which contains the relation of event instances to activity life cycle transitions $LT\hat{E}$. Second, the final event log L'' is constructed by clustering the event instances to activity instances yielding the relation of activity life cycle transition instances to event

7 The source code of the command line application can be downloaded from the subversion repository at https://svn.bpt.hpi.uni-potsdam.de/svn/EventActivityMatching-Evaluation/.

instance, L$\hat{\text{T}}\hat{\text{E}}$. We will first report the results for the derivation of L′. Here, the focus lies on assessing the feasibility and correctness of applying the concepts related to requirements R2 (Different abstraction levels), R4 (Shared functionalities), R5 (Missing events), and R6 (Additional events). Coming from the transformed event log L′, we turn to the creation of the final event log L″ by describing the clustering of event instances to activity instances. The focus, here, lies on the evaluation of the feasibility and correctness of the application of the defined concepts relating to requirements R1 (1:1 matching to activities), R3 (Loops and parallelism), R7 (1:1 matching to life cycle transitions), R8 (Hierarchical matching), and R9 (Nonconforming execution). To highlight the importance of a correct mapping, we furthermore evaluate the impact of different activity instance clustering settings on performance and conformance results.

As mentioned before, the incident management and the change management process of a large German IT outsourcing company are used in the conducted case study. Both processes are well documented with process models and work instructions and both processes are supported by the integrated ticketing software of IBM Tivoli Service Request Manager[8] and IBM Tivoli Change and Configuration Management Database[9]. The process model for the incident management process has 41 activities and the corresponding event log contains about 17,000 cases, 39 event classes and a total of about 550,000 event instances for a selected month. For the change process, the model contains 63 activities and the event log about 2,000 cases, 55 event classes and about 125,000 event instances. For both processes a number of traces were extracted from the original event logs to allow manual checking of the results by the responsible process managers. For the extract event logs, the variants with the highest number of traces were taken, while trying to ensure to keep most of the event classes. This resulted in a log with 401 traces for the incident management process and a log with 17 cases for the change management process.

Note that the activity life cycles for the activities of the two processes are not modeled and are therefore unknown. Still, it is possible for a domain expert to assign different event classes to different life cycle transitions of the simplified general activity life cycle model presented in Figure 7 in Section 2.1.7. The next section reports the results for the different steps and relates them to the requirements.

8 See http://www-03.ibm.com/software/products/en/servicerequestmanager.
9 See http://www-01.ibm.com/software/tivoli/products/ccmdb/features.html.

8.2.2 *Results for the Matching of Event Instances to Activity Life Cycle Transitions*

Starting with the first step of the base approach, the process managers of the two processes mapped all event classes to life cycle transitions of their corresponding activities. This essentially resulted in a mapping of event classes to activities, as almost all event classes where assigned to the placeholder transition ϕ. For both processes, only one event class was mapped to the suspend transition of its corresponding activities. For the incident management process, 28 out of 33 event classes have been mapped to 23 activities. Hence, five event classes could not be mapped to any activity and will have to be removed from the event log by assigning the removal placeholder. Since the incident process model contains 41 activities, there are 18 activities for which no event class could be assigned. Thus, these activities are turned into silent transitions for any further analysis like, e.g., conformance analysis. For the change process, 39 of the 42 event classes have been mapped to 31 out of the 63 activities. Thus, three event classes have to be removed and 20 activities are turned silent. In total, 113 relations between event classes and activity life cycle transitions (ϕ and suspend) have been derived for the change process and 56 relations for the incident process. In addition to these relations, for both processes event classes have been identified that potentially map to every activity (seven for the incident process and eight for the change process). For these event classes a mapping to the place holder *CLOSEST_ACTIVITY* have been added.

The results of the first step show, that we can handle missing events by identifying unmapped activities and turning them into silent transitions for further analysis, thereby fulfilling Requirement R5 (Missing events).

The next step in the base approach is to define context–sensitive mappings for those event classes that have been mapped to multiple activities, i.e., represent shared functionalities. In order to evaluate the disambiguation of shared functionalities requested by Requirement R4, the context conditions have been provided by the domain experts. During the first step, 20 shared functionalities have been identified for the incident management process. The change process contained 22 event classes that reflect shared functionalities. Out of these 22 event classes, eight have been mapped to the *CLOSEST_ACTIVITY* placeholder and therefore, do not need to be considered in this step. From the remaining 14 event classes, some belong to more than ten different activities. One example for such a shared functionality used by many activities is the communication protocol. The correspondence for these e-mails can be distinguished using attribute conditions over their headers. In total, there are 72 mappings that use attribute conditions for the change process. Another 42 mappings use a mixture

of attribute conditions and event context conditions. For the incident management process 39 attribute conditions and 11 event context conditions have been defined. These conditions again address 14 shared activities, which are very similar to those from the change process, as the underlying IT system is the same.

After defining the context–sensitive mappings to map event instances to activity life cycle transitions, there are still some event instances left that are not covered by any conditional mapping. As requested by Requirement R6 these event instances need to be removed. For those cases, conditional mappings to the placeholder *REMOVE_EVENT* have been defined (four for the change and three for the incident process).

With the defined context–sensitive mappings at hand, the third step of the base approach, the transformation of the event log, has been carried out. Table 24 lays out some basic statistics for the event log transformation for the incident management process. Coming from 33 event classes with 11,875 instances in the extracted event log of the incident process, the resulting event log of step three (L'_{Inc}) contains 11,375 event instances with 45 different event classes. The number of event classes increased due to the presence of shared functionalities and the mapping to two different life cycle transitions (ϕ and suspend). Note that an event class in this case is defined over the combination of the event name, i.e. the activity name, and the life cycle transition. Even though the number of event classes increased, the number of different trace variants decreased in the transformed event log. Furthermore, the number of event instances decreased as a result of the conditional and unconditional removals of events that could not be correlated with the process model.

The results for the change process are shown in Table 25. Here, a decrease from 42 to 36 in the number of event classes occurred. While the mapping for the change process contains the same number of shared functionalities and the same number of life cycle transitions, the decrease of event classes can be explained with the fact that for the change process there are more activities that are related to a larger number of event classes. Similar to the results of the incident process, the number of event instances decreased. Yet, for the change process, the number of trace variants remains stable.

The transformed event logs have been presented to the process managers and checked for correctness with respect to the mapping of event instances to activity life cycle transitions. While for the change process, a complete check was easily manageable, the transformed incident process log was only checked partially. The process manager was given one trace for each of the trace variants, i.e., 170 traces. For both processes, the checking returned no errors and we can thereby conclude the effectiveness of the base approach with respect to R2

Table 24: Event log statistics for the incident process.

	Complete month $L_{IncFull}$	Extract Raw L_{Inc}	Extract Mapped to LT $L'_{Inc}(L\hat{T}\hat{E})$	Extract Mapped to $\hat{LT}$ $L''_{Inc}(\hat{LT}\hat{E})$	Extract Mapped to $\hat{A}$ $L''_{Inc}(\hat{A}\hat{E})$
Event Classes	36	33	42	62	22
Cases	12,723	401	401	401	401
Variants	12,461	309	170	119	104
Event Instances	392,086	11,875	11,369	7,559	4,676
Life cycle transitions	–	–	2	3	1

(Different abstraction levels), R4 (Shared functionalities), R5 (Missing events), and R6 (Additional events).

Table 25: Event log statistics for the change process.

	Complete month $L_{ChgFull}$	Extract Raw L_{Chg}	Extract Mapped to LT $L'_{Chg}(L\hat{T}\hat{E})$	Extract Mapped to $\hat{LT}$ $L''_{Chg}(\hat{LT}\hat{E})$	Extract Mapped to $\hat{A}$ $L''_{Chg}(\hat{A}\hat{E})$
Event Classes	55	42	36	60	31
Cases	2,005	17	17	17	17
Variants	1,947	15	15	12	12
Event Instances	125,337	1,151	853	415	253
Life cycle transitions	–	–	2	3	1

8.2.3 *Results for the Activity Instance Clustering*

Having successfully evaluated the result of the first three steps, we turn to the final phase that automatically clusters the assigned event instances to activity instances. Here, we want to show that the base approach is able to fulfill requirements R1 (1:1 matching to activities), R3 (Loops and parallelism), R7 (1:1 matching to life cycle transitions), R8 (Hierarchical matching), and R9 (Nonconforming execution).

To this end, the domain experts first had to specify the correct instance borders. That is, they had to identify potential loops on inter and intra activity level. Both processes are supported by a form based web interface that allows for saving most of the fields edited by the user at any time. That means the user can enter text in a field, click the save button, enter more text, press save again, and so forth. For each saving, events are created. This means that for most event classes a sequence of events of the same event class is not truly a repetition on activity level. However, in case two different users edit a field, this must be considered as repetition establishing a new activity instance border. As this is the case for most of the event classes in the two processes, this instance border is defined for almost all mapping tu-

ples. Nevertheless, there are event classes where event repetition by different resources does not signal an activity repetition. Examples for this are status or progress events. Within one activity instance the status of the process instance might be set several times to waiting and back to working. Here, loops on the sub–activity level should not be lifted to activity level and thus, is has been declared that for these event classes no activity instance border exists.
For the assessment of Requirement R1 (1:1 matching to activities) and Requirement R7 (1:1 matching to life cycle transitions), the defined instance borders were used to generate two different final event logs for each of the two processes. For the process mining techniques that require a one-to-one mapping between events and activities on both type and instance level (R1), only the last event instance of each activity instance cluster was kept. This resulted in the event logs $L''_{Inc}(\hat{A}\hat{E})$ and $L''_{Chg}(\hat{A}\hat{E})$, for which Table 24 and Table 25 show general statistics. The event log for the incident process contains about 40 % of the event instances from the extracted raw event log and the number of variants has been reduced to one third. In contrast to that, the number of variants in the transformed event log for the change process is reduced by only 20 % while the number of events decreased to 22 % of the original number of events in the raw extract. Again, this is due to the fact that single activities in the change process are related to a higher number of event classes.

Turning to process mining techniques that require the mapping of events to activity life cycle transitions (R7), the two event logs L'_{Inc} and L'_{Chg} were transformed a second time. This time, all event instances mapped to a specific life cycle transition (i.e., the suspend transition) of an activity were kept. Furthermore, if mapped to the ϕ transition, the first and last event instances of each activity instance cluster were kept and assigned to the start and complete transition of the corresponding activity. This resulted in the two event logs $L''_{Inc}(\hat{LTE})$ and $L''_{Chg}(\hat{LTE})$. Due to the additional events for start and complete transitions, both event logs see an increase in the number of event classes, as displayed in Table 24 and Table 25 (62 event classes for the incident process and 60 event classes for change process). Nonetheless, the number of event instances and thereby also the number of trace variants decreases for both cases. This is due to the fact that there are many event instances that can neither be related to the life cycle transitions start or complete nor to the suspend transition.

All four event logs ($L''_{Inc}(\hat{A}\hat{E})$, $L''_{Chg}(\hat{A}\hat{E})$, $L''_{Inc}(\hat{LTE})$, and $L''_{Chg}(\hat{LTE})$) were checked for correctness of the mappings by the process managers. Again, for the incident process one trace per variant has been checked while the change process logs could be checked completely. All logs were approved to be correctly clustered.

For the purpose of demonstrating that Requirement R8 (Hierarchical matching) can be fulfilled, we also created the sublogs for each activity as explained in Section 3.3.4. For the event logs $L''_{Inc}(\hat{A}\hat{E})$ and $L''_{Chg}(\hat{A}\hat{E})$, all event instances of each activity instance cluster were saved into the corresponding activity sublog. The event classes of the original source events (from L_{Inc} and L_{Chg}) were assigned to those event instances. This allows to zoom-in on an activity and see the actual model of the raw events as produced by the IT system. Figure 47 shows the mined model from the sublog of the documentation activity of the incident management process. Here, the process analyst can gain further technical insides. The analyst may for instance detect that there is repetition of work on sub–activity level in some of the cases and that the summary field is not being updated for all of the cases.

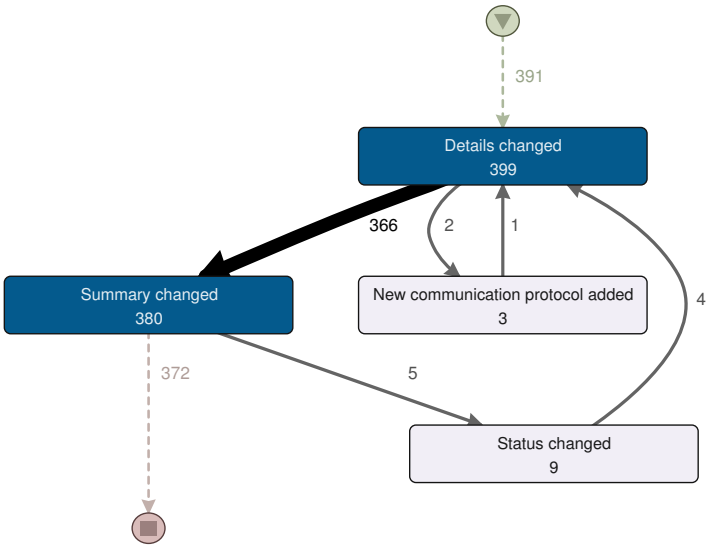


Figure 47: Mined subprocess model for the documentation activity of the incident management process (translated to English).

In order to show that the base approach is able to deal with nonconforming event logs as requested by Requirement R9, we assessed the conformance of the event logs $L''_{Inc}(\hat{A}\hat{E})$ and $L''_{Chg}(\hat{A}\hat{E})$ to prove that these are indeed not completely conforming to the designed models. The conformance to the designed process model has been measured using the constraint–relative behavioral profile conformance metric as defined in [138]. The incident management process returned a conformance metric of 63,79 % and the change management process has a conformance of 83,78 %. This shows that both processes contain nonconforming behavior. Yet, this does not impact the base approach.

For the evaluation of Requirement R3 (Loops and parallelism), we compare the clustering results for different activity instance border

definitions. Figure 48 shows the correct assignment of events to activity instances according to different definitions of activity instance borders. The very left bars show the results for the instance borders defined by the domain experts that were manually checked and found to be overall correct. The next best alternative to these manually defined instance borders is to define no activity instance borders at all, which means that event instances assigned to the same activity type are always clustered into one activity instance. This configuration yields around 90 % correctly assigned event instances for both cases. The explanation for this is that, on the one hand, the activities that are seen to be non-repeatable account for a large share of the event instances. On the other hand, it can be seen that there is not much repetition on activity level. Figure 48 also shows the results for the heuristic instance border that defines a maximal distance between two event instances belonging to the same activity instance. We have chosen the two configurations of a maximal distance of 1 and 2 as these are used by other abstraction approaches like in [59] and [75]. A distance of 1 means that there must not be any event of another activity in between two event instances belonging to the same activity instance, i.e. no concurrency. This configuration yields a fraction of around three quarters of correct event instance for the incident process, but only half of the event instances could be correctly assigned for the event log of the change process. The distance of 2 allows for one event instance in between and yields slightly better results. These results show that the handling of concurrency and loops needs attention and it shows that our approach is able to handle both, as requested by Requirement R3 (Loops and parallelism).

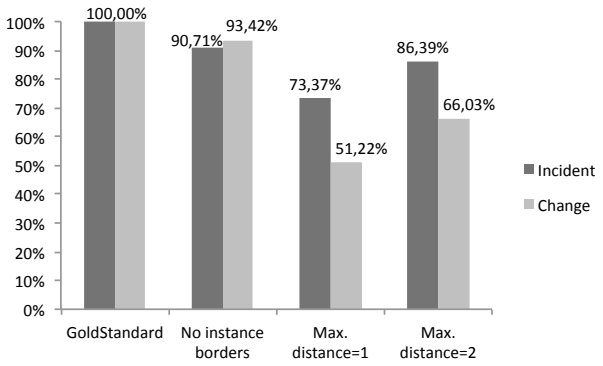


Figure 48: Fraction of correctly clustered event instances for different instance borders.

Finally, we investigated the influence of different instance border definitions on conformance and performance analysis in order to inspect how important a correct mapping is with regard to these analyses. We analyzed the constraint–relative behavioral profile conformance metric of the final event logs ($L''_{Inc}(\hat{A}\hat{E})$ and $L''_{Chg}(\hat{A}\hat{E})$) to the corresponding process model for the same definitions of activ-

ity instance borders as used before. The results of this analysis are presented in Figure 49. While the results for the incident process show only little variance, the conformance metric gives a maximal difference of around eight percentage points between the lowest and highest result for the change process. Thus, a considerable influence of the correct activity instance clustering on conformance checking can be observed.

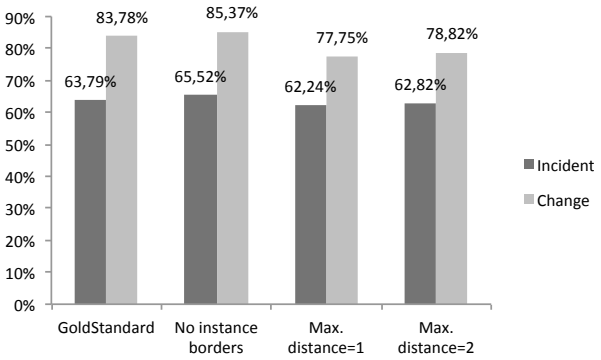


Figure 49: Conformance results for different instance border definitions.

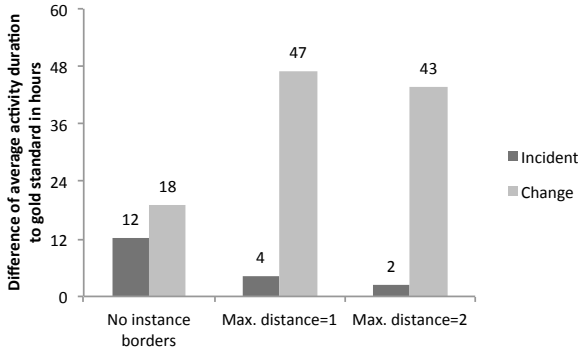


Figure 50: Differences in activity performance results for different instance borders in comparison to the gold standard.

Looking at performance analysis, the duration for each activity instance has been measured as the difference between its start and complete event. Figure 50 shows the difference of the average activity instance duration of the abstractions with the selected instance border definitions with respect to the gold standard. Here, the impact is even more striking with a maximum of almost two days of difference to the gold standard in the average duration. Moreover, a clear difference between the two processes can be seen. Again, differences are less high in the incident process. More interesting is that it can be clearly seen that for performance analysis there is a different ranking in the goodness of the different instance border definitions when comparing the results of the two processes. For the incident process the maximum distance between events yields the best result. In the per-

formance analysis of the change activities, no instance border turns out to be closer to the gold standard results in average. This shows how sensible these analyses are towards the right clustering.

8.2.4 *Summary and Discussion*

In this section, we provided an evaluation of the base approach by conducting two case studies with different processes from the ITIL context. Coming from the nine defined requirements, we assessed the feasibility and correctness of the different concepts introduced by the base approach. To this end, we could show that all nine requirements can be fulfilled in these real life scenarios. The produced results have been manually checked by domain experts and found to be correct.

Our work also adds an important perspective to the discussion of how quantitative results from conformance checking have to be interpreted. It has been emphasized in prior research that different definitions of conformance measurement yields significantly different values for the same process and log [54]. The fact that conformance measurement is also strongly influenced by the way how events are mapped and clustered to activity instances has received less attention so far. Our evaluation shows that different strategies of clustering events can yield strikingly different results (85 % versus 77 % for the change process). This finding emphasizes the importance of providing accurate techniques for matching events and activities.

Moreover, a correct abstraction will also benefit performance analysis on activity level, as the correct clustering of event instances is important to identify start and end of an activity to calculate the duration. Last but not least, process discovery also profits from a correctly abstracted event log. First, mined process models can be better understand by domain experts if they are on the abstraction level that is typically used in a company or department. Second, the introduced abstraction approach also allows for more advanced process discovery with the possibility to zoom into the different abstraction levels.

There are also limitations of the base approach. The accurate mapping of events to activity instances requires manual work. Our approach provides systematic support for automating a considerable share of this task. However, there is still a significant share of manual work required in order to encode missing domain knowledge into the required mapping definitions. This involves the definition of context conditions, event–activity relations, and instance border definitions. Still, these provide more accurate conformance and performance results as compared to techniques that cluster events without taking external knowledge into account as we have shown in our evaluation.

Looking at the generalizability of the base approach, we are confident that it can be used in any application scenario. Although our evaluation only looks at two specific cases, the approach itself is generic. Yet, the mappings are always domain specific, which means that you cannot simply transfer the mappings from one process to the next, but always need to create mappings that fit to the process and supporting application at hand. Nevertheless, more case studies should be performed in future research to provide a more stable ground for the evaluation of our concepts.

8.3 EVALUATION OF THE BEHAVIORAL APPROACHES

Building on the previously evaluated base approach, all behavioral approaches implement a semiautomatic means to retrieve the relation *Æ*, i.e., the type–level relation of activities and event classes. As the other parts of the approaches have been covered by the evaluation of the base approach in Section 8.2, this section only concentrates on the evaluation of the subtask of matching activity life cycle transitions to events on type level.

8.3.1 *Evaluation Goals*

The goal of the evaluation is to assess (1) the *effectiveness* and (2) the *efficiency* of the behavioral approaches. By effectiveness the ability to derive the correct mapping is meant. With efficiency, we refer to the necessary effort in terms of manual work. Furthermore, (3) the *robustness towards noise* and (4) the *performance* of the approaches shall be evaluated.

In order to measure (1) the effectiveness of the approaches, we evaluate whether the correct mapping can be retrieved within a reasonable time frame. Looking at (2), the efficiency, we quantify the manual work by counting the questions an analyst has to answer in order to arrive at the final mapping. The robustness towards noise (3) is evaluated by generating five different event logs for each process model with increasing levels of noise. For each process model, one event log with 1000 traces is simulated using the simulation technique provided by [98]. These noise-free event logs serve as a base to generate noisy event logs by randomly applying different noise patterns to a fraction of the traces. The noise patterns refer to shuffling, duplicating and removing of events. In this way, we produce four additional event logs for each process. One where noise is introduced into 25 % of the traces, and three more with 50 %, 75 %, and 100 % of noisy traces.

8.3.2 *Evaluation Setup*

The set of business processes used for the evaluation of our work on matching approaches based on behavior stems from the *BIT process library, Release 2009*, which has been analyzed by Fahland et al. in [51] and is openly available to academic research. The process model collection contains models of financial services, telecommunications, and other domains. The models are real life process models that have been anonymized to make them available for research. With hundreds of different processes, the library is well suited for the evaluation of our approaches based on log replay and behavioral relations. As the library only contains models but no event logs, we will simulate event logs using the simulation technique described in [98]. The used simulation technique assumes bounded Petri nets with an initial and a reachable final state and hence, imposes less restrictions on the process models than the behavioral approaches introduced in this thesis.

For the preparation and selection of the process models, the first step is to transform the models into Petri nets. During this transformation any disconnected activities are removed from the models. Having the connected Petri nets, further model checking is conducted to ensure only those models are kept that fulfill all assumptions made by our approaches. LoLA (a Low Level Petri Net Analyzer)[10] is used as a well-known model checker for Petri nets. The LoLA project, which exists since 1998, has been actively developed and was used in many case studies [147]. LoLA uses CTL* formulas to define and verify properties of a Petri net. CTL* combines LTL, which has been introduced in Section 2.1.6, and computation tree logic (CTL).

As required, all process models of the BIT process library have an initial state, i.e., an initial marking. The used simulation technique furthermore requires at least one reachable final state that is identified by a marking where all tokens are in places that do not have an outgoing transition. To filter out those models that do not have a final state, we derived the set of potential end places, i.e., $PL_{end} = \{pl | pl \in PL, |pl\bullet| = 0\}$. For each Petri net, we check whether a state can be reached where there are only tokens on potential end places and no tokens on other places. This is done using LoLA and the following formula:

$$\Diamond \left(\bigvee_{pl_{end} \in PL_{end}} pl_{end} > 0 \wedge \bigwedge_{pl_i \in PL \setminus PL_{end}} pl_i = 0 \right).$$

Both the simulation of process models and the generation of behavioral profiles require the Petri net at hand to be bounded. Boundedness of a Petri net can be checked with LoLA by proving that every place of the net is bounded. For a place pl this is done using the

10 See http://service-technology.org/lola/.

formula $AG\,pl < 00$. AG expresses that the formula has to globally hold on all paths, and $pl < 00$ stands for a boundary of arbitrary tokens on place pl. Note that we do not explicitly check for deadlocks other than the final state. Yet, all simulated event logs are checked to include events for all transitions. That is, we require each transition to be on an executable path from initial state to final state. Traces that do not end in a final state, because of a deadlock or a livelock, are excluded from the event log.

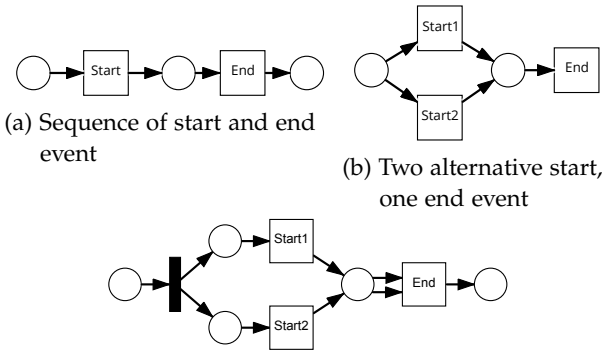
The BIT process library is separated into five groups of process models: A, B1, B2, B3, and C. Of these groups, B1, B2, and B3 contain different versions of the same models created at different points in time, with B3 entailing the latest versions [51]. Therefore, we only use the process models from groups A, B3, and C. In the further process of our evaluation we will not distinguish between these three groups.

Finally, we also removed all process models that only contain a single activity as matching is trivial for this case. After applying all of the described filtering steps, 442 models remain and are used for the evaluation of our behavioral approaches.

With respect to Requirement R1 (1:1 matching to activities) and Requirement R2 (Different abstraction levels), the evaluation of the behavioral approaches is divided into two parts, one for each of the two requirements. For each of the two parts, different event logs have to be generated. The event logs for the one-to-one matching were generated by simulating the process models using the techniques provided by Rogge-Solti [98].

In order to evaluate the handling of different abstraction levels, event logs were generated by simulating the process activities' enactment through event generators. Such event generators simulate a simple activity life cycle model containing a start and a complete life cycle transition. We chose three different event patterns that can be mapped to such a life cycle model based on the process instantiation patterns introduced by Decker and Mendling in [34]. Figure 51 depicts the different chosen patterns. Figure 51a shows a simple model with one start and one end transition, demonstrating a typical pattern found in many systems. For each activity assigned to this event model, a start and an end transition are generated for each execution of that activity. The second event model, depicted in Figure 51b, generates for each execution either an event "Start1" or an event "Start2" and always an end event. Thus, there are two alternative starts for such an activity, e.g., it could be started by an incoming mail or by a telephone call. The event model presented in Figure 51c also has two different start transitions, but in contrast to the model in Figure 51b, both start events always occur with no restriction on their order. For the simulation of the process models, each activity is randomly assigned to one of these three event models, or it is left as is, generat-

ing only a single event. Again, all generated event logs contain 1,000 traces and are limited to 1,000 events per trace as a stop condition for process models containing loops.



(a) Sequence of start and end event

(b) Two alternative start, one end event

(c) Two concurrent start, one end event

Figure 51: Different event models used to generate events.

All experiments were conducted in a cluster environment where each matching experiment was assigned six gigabytes of main memory and four CPU cores with 2.93 GHz. This reflects the processing power of a typical desktop machine these days. For each experiment a timeout of ten minutes had been set, after which the experiment was terminated if the constraint satisfaction problem was not yet solved.

8.3.3 *Results for the One-to-one Matching of Activities and Events*

8.3.3.1 *Replay Approach*

Starting with Requirement R1 (1:1 matching to activities), we first focus on *(1) the effectiveness* of the replay approach. To this end, we inspect the number of correctly matched cases. We distinguish between three categories: "correctly solved", "constraint conflict", and "resource shortage". The latter category points to the cases where the constraint satisfaction problem could not be solved because of a lack of main memory or too long execution time. The category "constraint conflict" summarizes the cases where too many incorrect constraints exclude the correct mapping from the solution space. Hence, the approach does not prove to be effective for these combinations of process model and noise level in the execution log.

From Figure 52 one can see that the mapping for 93 % of the processes with noise-free logs can be solved correctly, while the remaining 7 % cannot be mapped due to resource shortage. When every fourth trace in the event logs contains noise (25 %), the replay approach runs into resource shortage a bit more often and fails due to constraint conflicts in one case. This leads to an effectiveness rate of 92 % for this noise level. With increasing noise, the percentage of unsolved cases increases slightly to 9 % for 50 % noisy traces and to 11 %

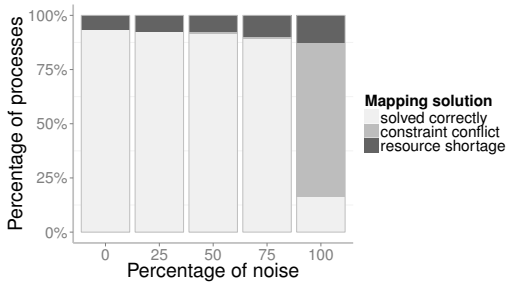


Figure 52: Replay approach: Solved and unsolved matchings.

for event logs with 75 % of traces with noise. Yet, the fraction of unsolved or incorrectly solved mappings is still rather low for mappings of event logs with a noise level below 100 %, yielding a good effectiveness of 89–93 % of correctly solved mappings. For the event logs where all traces contain noise, this cannot be said. Figure 52 depicts that the vast majority (83 %) of cases cannot be solved correctly for a noise level of 100 %. Here, the main reasons are conflicting constraints caused by the high noise level in the event logs.

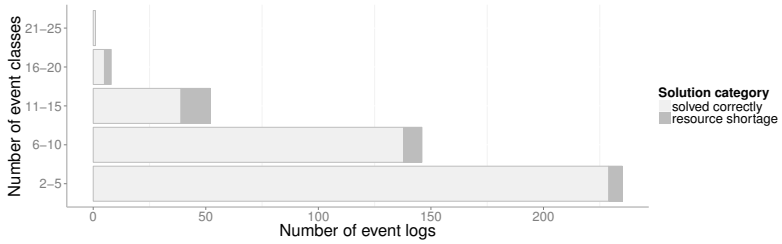


Figure 53: Replay approach: Solved and unsolved matchings by number of event classes.

Figure 53 looks at the efficiency of the replay approach from another angle by inspecting the influence of the number of event classes that are contained in an event log. We therefore build five categories. For small numbers of up to five event classes per log the replay approach is able to solve 97 % of the matchings without running into resource shortage. For the event logs with six to ten event logs the percentage of unsolved matchings increases to 5,5 %, leading to an effectiveness rate of 94.5 %. In the next category of event logs with eleven to fifteen event classes the effectiveness drops further down to 75 %. When mapping event logs with 16 to 20 event classes, the replay approach is only effective in two thirds of the cases. The last category holds only one event log that entails 21 event classes and can be correctly matched. Yet, it is hard to draw conclusions for these last two categories as there are only nine event logs for these categories. Nevertheless, the trend becomes obvious. With a higher number of event classes the replay approach is more likely to run into resource

shortage. Albeit, it is still successful for the event log with the highest amount of event classes.

A closer inspection of the process models from the failed matchings revealed that a high degree of concurrency is another factor that may lead to resource shortage. In some of the models all activities are concurrent. For these cases the replay approach could not be of help even if the CSP could be solved with the available resources.

With reference to the evaluation of *(2) efficiency*, we measured the number of questions that had to be asked in order to arrive at the final mapping. When the mapping is performed completely manually, the number of questions is equal to the number of event classes minus one, as one has to specify the corresponding activity for each event class until there is only one event class and one activity left for which the mapping is clear. Taking this into account, Figure 54 shows how

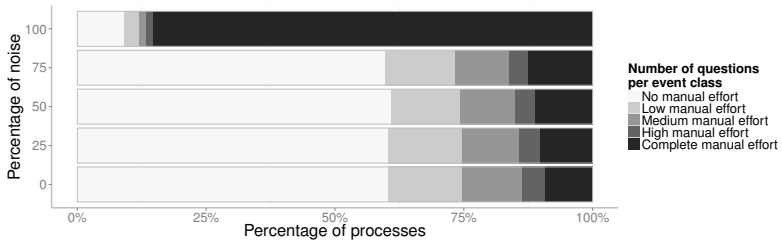


Figure 54: Replay approach: Number of questions per event class for each noise level.

many questions needed to be asked per event class for the different levels of noise in the event logs when using the replay approach. This relation is calculated as $\frac{q}{|E|-1}$, where q is the number of questions and E the set of event classes in the given log. Note that we subtract one from the number of event classes, as it is not necessary to ask any question if there is only one event class left. The results are clustered into five groups. The first group includes the cases where no manual effort was required ("No manual effort"). These cases account for 65 % of all mappings for event logs with a noise level below 100 %. The second category is "low manual effort", which is defined by the interval $(0, 0.25]$. This category includes all cases where we only had to ask questions for less than every fourth event class. We consider this as low effort. It can be seen that 14–15 % of the cases with a noise level below 100 % fall into this category. Another 11–13 % of these cases account for the category of "medium manual effort", which we define by the interval $(0.25, 0.5]$. Here, at most for half of the event classes we need to ask a question. Next come the cases with "high manual effort", for which we needed to ask a question for more than every second event class but not for all, i.e., the interval $(0.5, 1)$. With 4–5 % of the cases with 75 % or less noisy traces, this fraction is rather small. Finally, there is a fraction of 9–12 % of all tested event

logs where not all cases contain noise, for which the approach did not help. Here, a question needed to be asked for every but the last event class ("Complete manual effort"). In total, 90–93 % of all cases that contain at most a fraction of 75 % of noise-free traces can be handled with at most medium effort. Moreover, almost two third of all cases with a noise level below 75 % can be handled fully automatically without asking any question.

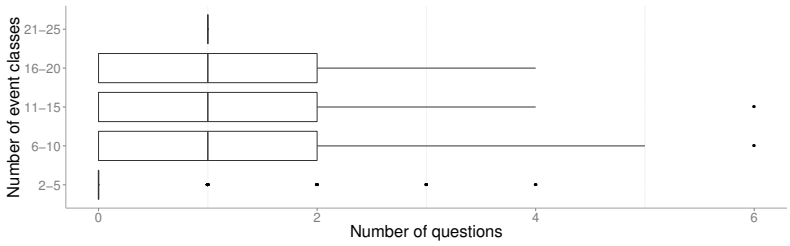


Figure 55: Replay approach: Number of questions depending on the number of event classes for correctly matched event logs without noise.

Figure 55 gives another view on the number of questions per event class. Here, we inspected all event logs without noise and divided the event logs by their number of event classes into five categories. It can be seen that for event logs with two to five event classes, the approach typically does not need to pose any question. For all other cases we do not need to ask any question for a quarter of the processes and 50 % of all matchings could be solved with at most one question. The highest number of questions that had to be asked is six questions. Interestingly, one can see that the number of questions does not increase with the number of event classes.

While evaluating (1) the effectiveness and (2) the efficiency, we already included the results for the different levels of noise in order to assess *(3) the robustness towards noise*. Summing these results up, it can be seen in Figure 52 that the approach is very robust towards noise for all tested levels below 100 % of noisy traces. Only for event logs where all traces contain noise, the approach does not work in the majority of cases.

Looking at the *(4) performance* of the replay approach, Figure 56 depicts the duration of the matching with respect to the number of event classes involved. When measuring the performance, we consider only cases that have been correctly solved, as the complete time for unsolved matchings is unknown due to the set timeout. The duration includes the setup of the CSP as well as the complete simulated questioning. Hence, the duration shown in Figure 56 potentially includes multiple solution runs of the constraint satisfaction problem solver. It can be seen that the duration of the matching drastically increases

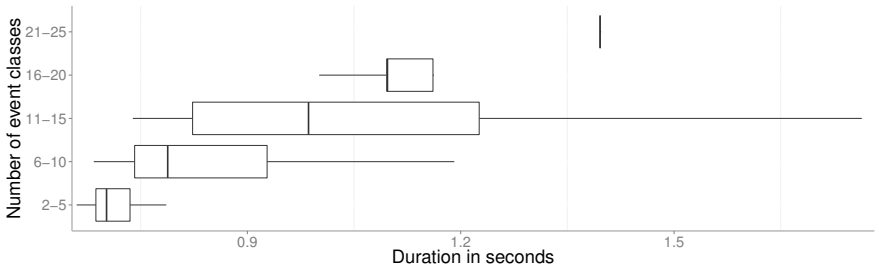


Figure 56: Replay approach: Duration of the matching depending on the number of event classes for event logs without noise (without outliers).

with growing numbers of involved event classes. Still, even for event logs with 16–25 event classes, the complete matching takes less than 1.2 seconds. For event log with the largest number of event classes, the matching is completed in 1.4 seconds. The overall maximum lies at 18.9 seconds for an event log in the category of 11–15 event classes.

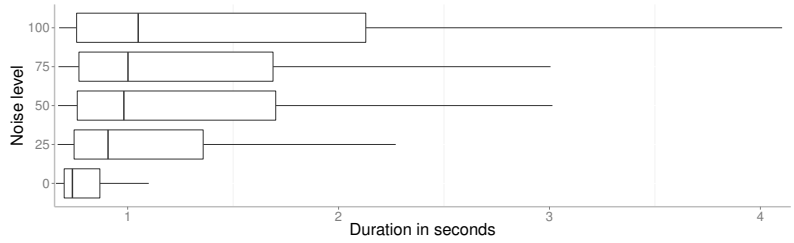


Figure 57: Replay approach: Duration of the matching depending on the noise level of the event log (without outliers).

Figure 57 inspects the performance of the replay approach with respect to the inserted noise in the event logs. It can be seen that the durations of the matching increases with increasing noise level. While we do not see quite a drastic difference between the medians of the different noise levels as for the categories of event classes, a significant decline in performance from no noise to some noise can be seen. Even for event logs where all traces contain noise, 75 % of the matchings can be solved in less than 2.25 seconds. We believe that this is reasonably fast since the matching is a one-time undertaking that is hardly time-critical.

8.3.3.2 *Behavioral Profile Approach*

Turning to the behavioral profile approach, we start with the definition of different configurations for the approach. We define a *basic* configuration that uses none of the optional constraints and no relaxed constraints. Hence, the basic configuration uses no constraints for interleaving or direct follower relations, no relaxed strict order

and no relaxed co–occurrence. Next, we define five different configurations that are all based on the basic setting. The *direct followers* configuration adds the constraints derived from direct follower constraints. Constraints stemming from interleaving relations are added in the *interleaving* configuration. The *relaxed co–occurrence* and *relaxed strict order* configurations use the relaxed definition of the respective constraints. The last configuration is a combination of the already defined configurations but leaves out the interleaving constraints. Hence, it is called *all but interleaving*.

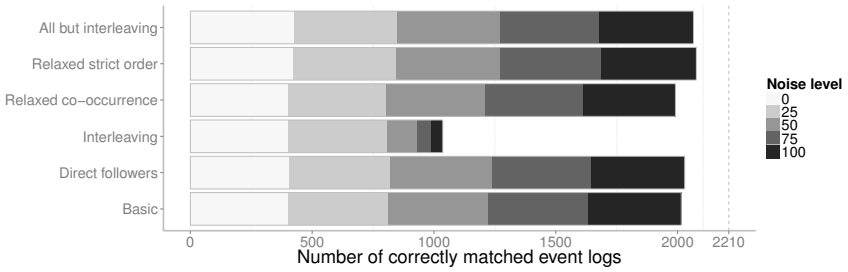


Figure 58: Behavioral profile approach: Number of correctly solved matchings in one-to-one setting for different noise levels.

Assessing *(1) the effectiveness*, Figure 58 depicts the number of correctly solved matchings for the six different configurations for each noise level. First of all, it can be seen that for almost all of the configurations the mapping is correctly solved for around 2000 event logs. That is, 90–94 % of all mappings are solved correctly for these configurations. Only the *interleaving* configuration has major problems and solves only around 47 % correctly. When breaking the results down to the different noise levels of the event logs, it can be seen that the *interleaving* configuration has severe problems when dealing with noise levels above 25 % of noisy traces per log. In contrast, all other configurations are rather stable towards noise.

Looking only at event logs without noise, the *relaxed strict order* and the *all but interleaving* configuration score highest with 96 % correctly solved mappings. All other configurations only reach 91 % in effectiveness for noise-free event logs.

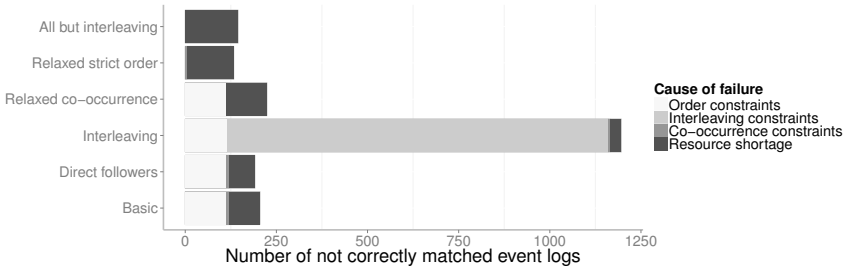


Figure 59: Behavioral profile approach: Number of not correctly matched event logs (all noise levels).

Figure 59 sheds light on the reasons for not correctly solved mappings. We therefore drill down to the specific types of constraints that were pushed into the constraint satisfaction problem. Looking at the *interleaving* configuration, it can be seen that for more than a thousand event logs, wrong constraints were derived from discovered interleaving relations. As known from Figure 58, the *interleaving* configuration has problems with noisy logs. Figure 60 depicts how the number of constraint conflicts evolves for the *interleaving* configuration with increasing noise. It can be seen that only the constraints stemming from interleaving relations fuel the increase in conflicting constraints. There are no incorrect interleaving constraints for logs without noise and only few matchings suffer from misleading interleaving constraints when 25 % of all traces are noisy. With half of the traces containing noise, there is a huge increase in matchings with incorrect interleaving constraints. Already 300 out of 442 (68 %) event logs cannot be matched anymore due to interleaving constraints. At noise level 100, 377 of all 442 matchings (85 %) contain wrong interleaving constraints. In contrast to this, the conflicting order constraints decrease with higher noise levels. The reason for this is that with increasing noise, less order relations reach the minimum support and therefore less order constraints are created. These relations that are not seen as order relations anymore, are then seen as interleaving relations and result in constraint conflicts.

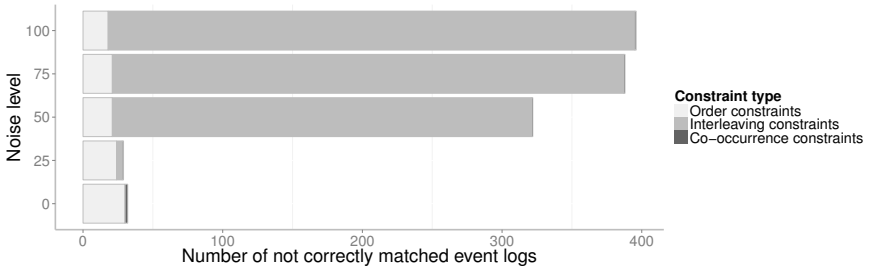


Figure 60: Behavioral profile approach: Number of matchings with wrong constraints by noise level for the setting with interleaving constraints.

The second main reason for conflicting constraints for the behavioral profile approach are interleaving activities for which their corresponding events show a dominant ordering. For all but the top two configurations, around 5 % of the matchings cannot be solved correctly due to wrong constraints stemming from strict order relations. These wrong constraints can be resolved by employing the relaxed mapping for strict order relations. Hence, the *relaxed strict order* and *all but interleaving* constraints do not contain any incorrect order constraints.

In a similar way as the order constraints, also the co–occurrence constraints suffer from dominant behavior in the event log. Here, the

root cause lies in optional activities that are executed in a dominant fashion, i.e., are present in almost all cases. In the simulated data this happened only for the event logs of one process and is resolved by using relaxed co–occurrence constraints.

Turning the CSP into an optimization problem does not help to overcome constraint conflicts in most of the cases because all of the processes with constraint conflict result in a resource shortage when using an optimization problem. As for the replay approach, resource shortage relates to timeouts and memory shortage. The solution of the optimization problem requires often too high computational efforts. Therefore, we will focus on the results obtained without using an optimization problem.

Even without using an optimization problem, Figure 59 reveals that there are cases that cannot be solved due to resource shortage. It can be seen that the number of cases with resource shortage decreases when additional constraints from interleaving or direct follower relations come into play. On the contrary, the number of cases with resource shortage increases when constraints are relaxed. While not using or relaxing certain constraints removes conflicts that prevent the correct mapping, it comes at the price of higher computational effort as the search space grows. If a process can be solved or not with a certain configuration heavily depends on the structure of the process and on the characteristics of the event log. As for the replay approach, processes with a high degree of concurrency often lead to resource shortage.

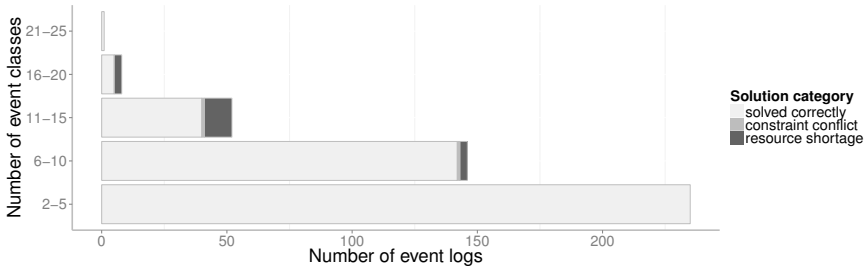


Figure 61: Behavioral profile approach with relaxed strict order constraints: Solution categories by number of event classes for event logs without noise.

Figure 61 provides another perspective on the effectiveness by relating the number of correctly matched and not correctly matched event logs to the number of event classes in the event log. Here, we focus on those event logs without noise for the configuration with relaxed strict order constraints, which scored best in terms of effectiveness. It can be seen that event logs with up to five event classes can always be solved correctly. For event logs with six to ten event classes a small fraction of 2 % cannot be solved due to resource shortage. This fraction increases and peaks for event logs with eleven to fifteen

event classes. For these event logs about one fifth of the matchings terminate due to resource shortage. In the next bigger category already one third of the mappings cannot be constructed because the approach takes too long or runs out of memory. The one event log with 21 event classes that is contained in the last category can be correctly solved. This proves again that, while the number of event classes does have some effect on effectiveness, it cannot be said that one can clearly predict from the number of event classes whether a mapping will run into resource shortage or not.

Summing up the results for (1) effectiveness, we could show that the behavioral profile approach performs very good with 91–96 % of correctly solved mappings for most of the configurations when event logs are noise–free. Over all noise levels, still, 90–94 % of the event logs can be solved correctly. Thereby, the behavioral profiles approach almost always slightly outperforms the replay approach in terms of effectiveness.

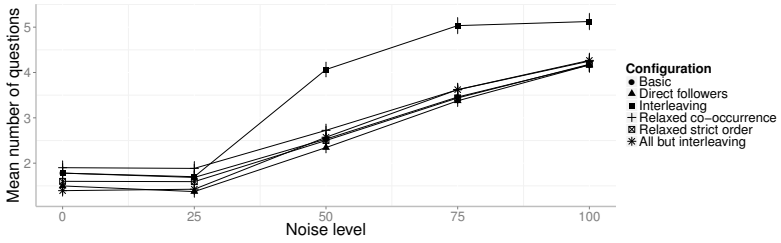


Figure 62: Behavioral profile approach: Mean number of questions for each configuration.

In order to assess *(2) the efficiency* of the behavioral profile approach, we measured the mean number of questions that had to be asked for each configuration for each noise level, as depicted in Figure 62. For noise levels zero and 25 %, the data shows that all configurations result in a similar mean number of questions ranging from 1.4 for *all but interleaving* to 1.78 for the *basic* configuration. For all noise levels above 25 %, the number of questions increases for all configurations. Still, almost all configurations behave very similarly with a steady increase of one question on average. Only the *interleaving* configuration requires significantly more questions. From the results of (1) the effectiveness, it is known that the *interleaving* configuration is not able to solve many mappings for event logs with noise levels above 25 %. For those cases the maximum number of questions, i.e., one question for each event class, has to be asked.

Figure 63 provides deeper insights into the distribution of the required questions by providing the corresponding box plots for each noise level. Focusing first on the box plots for the event logs that do not contain any noise, it can be seen that the *direct follower* and the *all but interleaving* configurations differ from all other configurations by

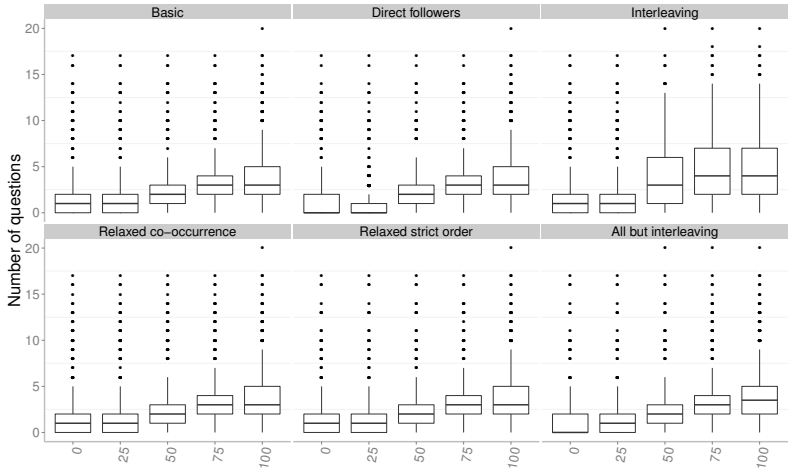


Figure 63: Behavioral profile approach: Boxplots showing the number of required questions for each noise level for each configuration.

yielding a median number of questions of zero. That is, no questions have to be asked for half of these cases, while all other configurations require up to one question for half of the event logs without noise. This is due to the fact that the direct follower relations enable meaningful constraints for processes with large loops. Therefore, both configurations require less questions for these types of processes.

In order to test statistical significance of these results, we employ statistical tests. As the number of questions do not follow a normal distribution, the Mann-Whitney test is used [146, p.139f]. The tests show that there is a significant difference (p-value < 0.05) for all other configurations and the *direct follower* relation. Yet, there is no significant difference between the *all but interleaving* configuration and any other configuration. Hence, the *all but interleaving* configuration lies in the middle between the *direct follower* configuration and all other configurations, having no significant difference to either of both sides. This is explained by the fact that although the *all but interleaving* configuration uses direct follower relations, it also uses the relaxed constraints for co–occurrence and strict order, which in some cases lead to more questions.

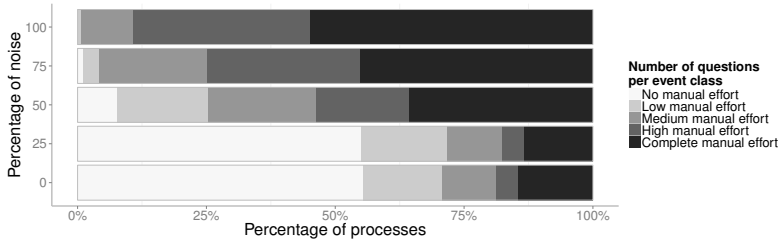


Figure 64: Behavioral profile approach with direct follower relations: Number of questions per event class for each noise level.

As the *direct follower* configuration showed the best overall performance in terms of efficiency, we focus on the results of this configuration for further investigations. Figure 64 presents the number of required questions in relation to the number of event classes for the *direct follower* configuration. For the event logs where less than every second trace contains noise, the behavioral profile approach with direct follower relations is able to handle 55 % of these cases completely automatically without asking any question. This is significantly less than the replay approach could handle without questions (65 %). For 15–17 % of the cases with a noise level below 50 %, the behavioral profile approach required low effort by the analyst (up to one question for every fourth event class). Another 10–11 % could be handled with medium effort (up to one question for every second event class). In total, the behavioral profile approach is able to provide the matching for 80-83 % of the cases with less than 50 % noisy traces with at most medium effort. Again, this is 10 percentage points less than the replay approach could handle with medium effort (90–93 %). What is more, Figure 64 reveals that the behavioral profile approach with direct follower constraints is more often not helpful as it has to pose one question for each event class for 13–14 % of the event logs with less than 50 % of noisy traces. Interestingly, this number is lower for event logs with 25 % of noisy traces than for event logs without any noise. This is due to the fact that the number of conflicting order constraints decreases with increasing noise levels as seen in Figure 60 for the interleaving configuration. Therefore, more mappings can be solved and the approach can be of help to the analyst.

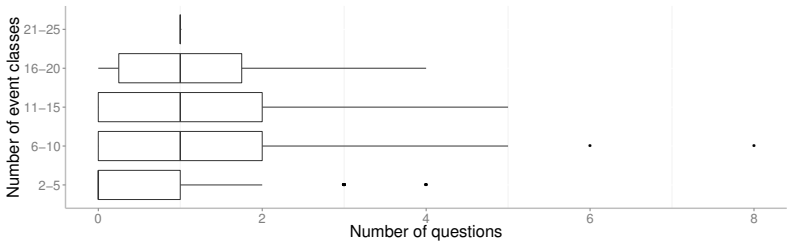


Figure 65: Behavioral profile approach with direct follower relations: Number of questions per event class for correctly matched event logs without noise.

Figure 65 depicts the box plot view of the number of questions per event class category for the behavioral profile approach with relaxed mapping for strict order relations. In the same way as in Figure 55 for the replay approach, the plot only contains event logs without noise. Event logs with very few event classes (2–5) can be solved completely without questions in half of the cases, whereas the fraction of automatically solved mappings was three quarter for the replay approach. Equally to the replay approach, half of the event logs with more than five event classes can be solved with only one question and three

quarters can be solved with at most two questions. Yet, the maximum number of questions increased from six for the replay approach to eight. Overall, the behavioral profile approach with direct follower constraints performs almost equally efficient as the replay approach but is less efficient for some of the event logs especially with a low number of event classes present in the logs.

Beyond efficiency, Figure 64 informs about *(3) the robustness towards noise* of the behavioral profile approach with direct followers. When comparing these results to those of the replay approach, it becomes obvious that the behavioral profile approach is less stable towards noise. Looking at event logs with 50 % and more noisy traces, the overall picture of required questions per event class drastically declines, as Figure 64 reveals. Already with a noise level of 50 %, the number of processes that can be handled without any question significantly drops down from 55 % to 8 %. This number further declines to 3 % for a noise level of 75 %. When all traces contain noise, the behavioral profiles approach can handle only 10 cases without asking any question. The number of cases with high or complete manual effort increase almost with the same rate with which the number of cases with no manual effort decreases. When half the traces in the event logs contain noise, the behavioral profile approach poses one question for each event class in 36 % of the cases. This number increases to 45 % for a noise level of 75 % and to 55 % when all traces are noisy. Nevertheless, the behavioral profiles approach outperforms the replay approach for event logs where every trace contains nonconforming behavior. While the replay approach is only helpful for 14 % of those cases, the behavioral profile approach with direct follower constraints eases matching for 45 % of those event logs.

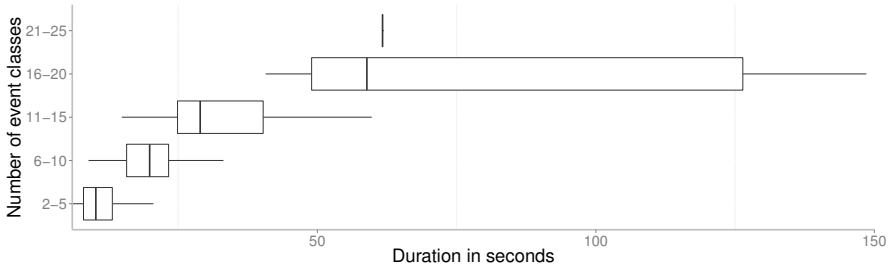


Figure 66: Behavioral profile approach with relaxed mapping for strict order relations: Duration of the matching depending on the number of event classes (without outliers).

With respect to *(4) performance*, Figure 66 provides the runtime of the matching with direct follower relations. It can be seen that already for event logs with up to 5 event classes the mapping takes up to 10 seconds for half of the cases and 20 seconds at maximum. Here, the

replay approach always requires less than a second. One can also see that there is an even higher difference in performance with increasing number of event classes as we saw for the replay approach. For 50 % of the event logs with 16–25 event classes, the matching takes up to one minute. The maximum duration lies at 2.5 minutes.

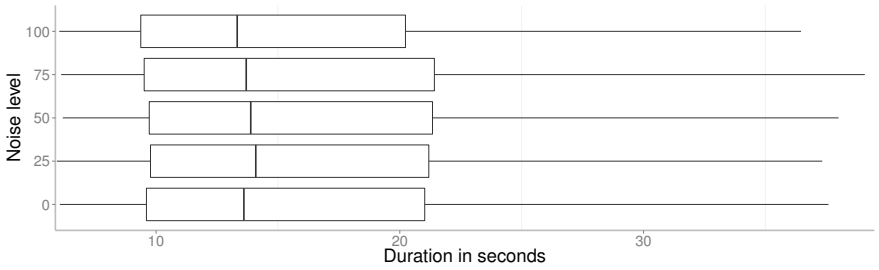


Figure 67: Behavioral profile approach with direct follower relations: Duration of the matching depending on the noise level (without outliers).

Figure 67 shows the performance of the behavioral profile approach with respect to the different levels of noise in the event logs. In contrast to the replay approach, no significant difference between noise-free and noisy logs is detected. Thus, for the cases that can be correctly solved, the performance of the behavioral profile approach remains stable even with increasing noise. While the replay approach is significantly faster, we believe the behavioral profile approach is still fast enough, as the matching on type level is only a one-time endeavor.

8.3.3.3 *Declare Approach*

From the results of the behavioral profile approach, we now turn to the Declare approach. Similar to the behavioral profile approach, we start with a basic setting that does not include constraints from interleaving relations and does not use the relaxed mapping for strict order and co–occurrence relations. Building on this configuration, almost the same configurations are defined as for the behavioral profile approach. Nonetheless, there are some differences in the configurations for the Declare approach. First, there is no direct follower configuration for the Declare approach, because direct follower relations are only indirectly used by the Declare approach in the basic ordering relations. Second, there is an additional configuration that builds on the basic configuration and uses the *alternative participation* constraint. The *all but interleaving* configuration again uses all introduced relaxed constraints, i.e. relaxed strict order and relaxed co–occurrence. Furthermore, it does not use interleaving constraints and it sticks to the base participation constraint for mandatory events and activities.

Starting with *(1) the effectiveness* of the Declare approach, Figure 68 shows the number of correctly matched event logs for each of the six

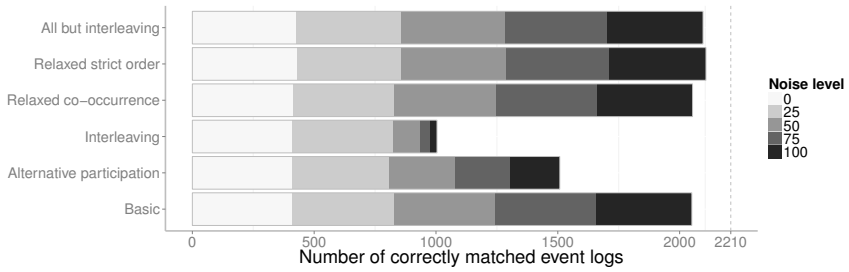


Figure 68: Declare approach: Number of correctly solved matchings in one-to-one setting for different noise levels.

configurations and for each noise level. While the results depicted in Figure 68 look similar to those previously shown for the behavioral profile approach, small differences can be identified. In all configurations, except for the *interleaving* and *alternative participation* configuration, the Declare configuration is able to solve about one to three percentage points more mappings correctly. This results in an overall effectiveness rate of 93–95 % for these configurations. The newly introduced configuration *alternative participation* seems to have similar problems with noise levels above 25 % as the *interleaving* configuration. Yet, it does perform better on all three levels above 25 % of noisy traces in each log. Overall, about 1500 event logs are mapped correctly using the *alternative participation* configuration, which is about 68 % of all event logs. Focusing on the effectiveness for noise-free event logs both the *relaxed strict order* and the *all but interleaving* configuration match 97 % of all event logs correctly on type level, with the former configuration being a bit ahead when it comes to the total counts. All other configurations yield a share of 93 % correct mappings for event logs without noise.

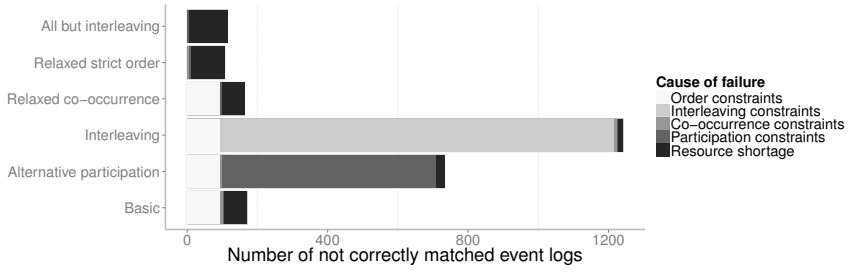


Figure 69: Declare approach: Number of not correctly matched event logs (all noise levels).

In Figure 69, we inspect the root causes for not correctly matched event logs by enumerating failed matchings with wrong constraints or resource shortage. Starting with the latter, the same pattern for resource shortage that has been identified for the behavioral profile approach can be seen for the Declare approach. Adding constraints that lead to no solution speeds up the matching and thus, results in

fewer cases of resource shortage. This is the case for the *interleaving* and for the *alternative participation* configuration. Relaxing order constraints, in contrast, enlarges the search space and leads to resource shortage more often, as seen for *relaxed strict order* and *all but interleaving*, which contains the former. The relaxation of co–occurrence constraints does not increase the cases with resource shortage. This shows that the impact of ordering constraints is higher than the impact of co–occurrence constraints in the Declare approach. Overall, it can be that there are fewer cases of resource shortage for the Declare approach than for the behavioral profile approach. This can be accounted for by the additional constraints that the Declare approach introduces and that are not present in the behavioral profile approach, namely, those constraints coming from existence rules. Nevertheless, processes with high concurrency still tend to fail due to resource shortage.

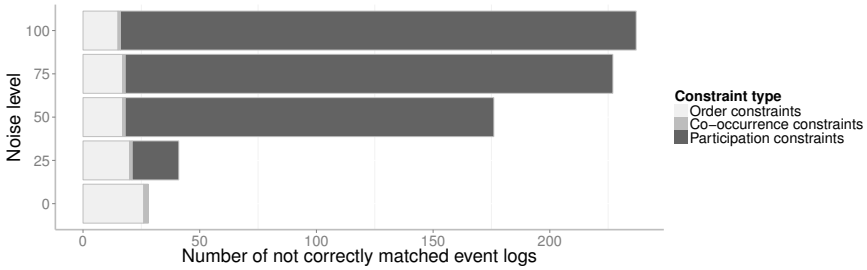


Figure 70: Declare approach: Number of matchings with wrong constraints by noise level for the setting with alternative participation constraints.

Turning to the inspection of incorrect constraints, we see a bit more than 4 % of the cases that do not use the relaxed strict order constraint failing due to wrong order constraints. This is similar to the results of the behavioral profile approach. Again, there is also one process for which the event logs cannot be mapped due to an incorrect co–occurrence constraint, which is a result of the dominant occurrence of the event belonging to an optional activity. For the declare approach, this also leads to an incorrect participation constraint for all configurations that use the base participation constraint. Yet, in contrast to the relaxed co–occurrence constraint, the alternative participation constraint leads to a steep increase in constraints that exclude the correct mapping. Figure 70 inspects how the number of wrong constraints evolve with increasing noise levels for the *alternative participation* configuration. As for the behavioral profile approach, we see a decreasing amount of incorrect order constraints with increasing noise because many order relations drop out with a too low support. The co–occurrence constraints are again not influenced by noise. In contrast to this, the wrong alternative participation constraints heavily increase with additional noise. With no noise, there are no incorrect

alternative participation constraints. For a noise level of 25 %, there are few matchings that fail due to wrong participation constraints. Yet, with half of the traces containing noise, 36 % (158 of 442) of all matchings fail due to misleading alternative participation constraints. This number rises to 50 % (221 of 442) for event logs where all traces contain noise. The reason for this steep increase is that events belonging to mandatory activities are too often skipped in event logs with high noise. Therefore, these events are seen as not mandatory and are incorrectly used for the matching by the alternative participation constraint, which basically enforces the matching of optional events to optional activities.

Constraints based on interleaving relations show similar effects for the Declare approach than those effects we saw for the behavioral profile approach. For no noise level and the lowest noise level, there is no difference in the effectiveness results. With higher noise levels, constraints from interleaving relations create constraint conflicts for the majority of cases.

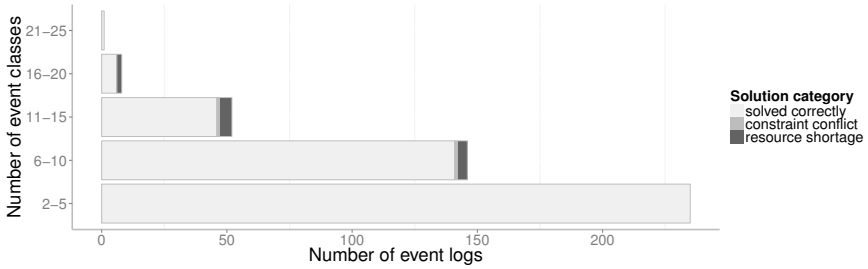


Figure 71: Declare approach with relaxed strict order constraints: Solution categories by number of event classes for event logs without noise.

Looking at the effectiveness of the Declare approach from the number of event classes that are contained in the matched event logs, Figure 71 presents the results for the configuration with relaxed strict order constraints, which scored best in the overall effectiveness provided by Figure 68. As for the previously discussed approaches, we focus on the event logs without noise to limit further side–effects. In contrast to the replay and the behavioral profile approach, the Declare approach is able to successfully provide the mapping for all event logs with up to five event classes. Thereby it is on par with the replay approach for this category. For the next two categories the effectiveness rates are 96.6 % and 88.5 %. When matching event logs with 16 to 20 event classes, the Declare approach with relaxed strict order constraints solves six out of eight matching problems. Again, the event log with 21 event classes is also mapped correctly. Overall, the Declare approach with the relaxed strict order configuration outperforms both the replay and the behavioral profile approach on

effectiveness for event logs without noise.

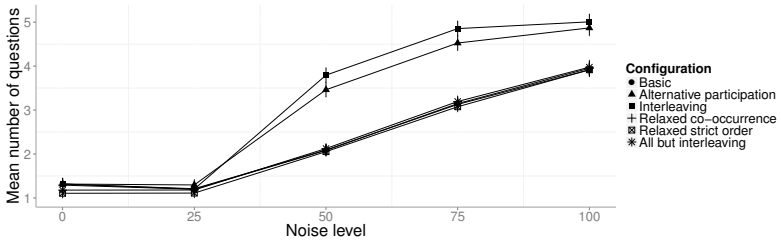


Figure 72: Declare approach: Mean number of questions for each configuration.

With Figure 72, the focus is changed to *(2) the efficiency* of the Declare approach. Again, the figure appears to be almost identical to the one seen for the behavioral profile approach. Yet, there are some differences. First, there are now two outliers when it comes to the required questions for event logs with a noise level above 25 %. Besides the *interleaving* configuration, also the *alternative participation* configuration requires significantly more questions on higher noise levels than the other configurations. This is in line with the previously discussed results for the effectiveness. As both configurations fail in a large set of matchings with higher noise, these cases decrease the mean efficiency as one needs to ask the maximum number of questions when no solution can be found. The mean number of questions for noise levels zero and 25 is slightly above one question, ranging from 1.18 for *all but interleaving* to 1.32. For all but the two outlier configurations, the mean number of questions increases roughly by one for each subsequent noise level. For the *interleaving* and *alternative participation* configuration, the mean number of questions goes directly up to about four questions for a noise level of 50 %. From 50 % to 75 % of noisy traces the slop of the curve is the same as for the other configurations, i.e., an increase of one question on average. From 75 % to 100 % there is only a very slight increase to 4.5 / 5 questions on average. Comparing these numbers to those of the behavioral profile approach reveals that the Declare approach performs slightly more efficiently. Applying the Mann-Whitney test to the overall numbers of questions for both approaches shows that this difference is also statistically significant.

Having compared the number of questions from the Declare approach with those of the behavioral approach, we want to see whether there are further differences between the different configurations of the Declare approach. Figure 73 depicts the box plots of the number of questions for each noise level for every selected configuration of the Declare approach. It can be seen that the distributions on noise levels zero and 25 are almost identical. The Mann-Whitney test also does not return any statistical significant difference between the con-

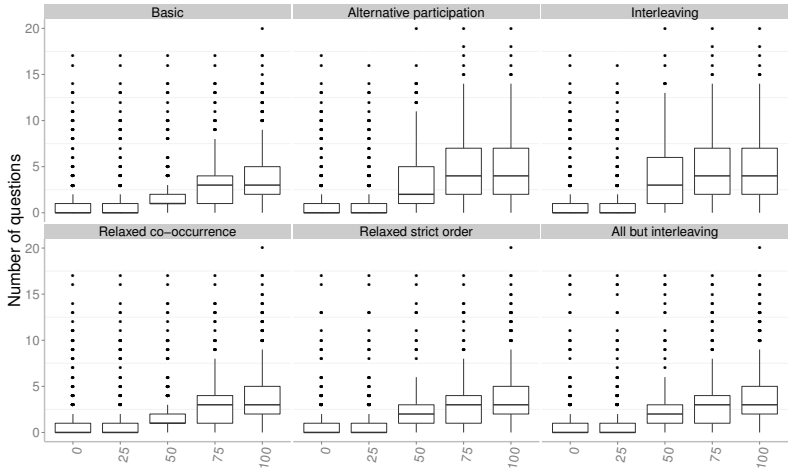


Figure 73: Declare approach: Boxplots showing the number of required questions for each noise level for each configuration.

figurations for these two noise levels. Looking at the higher noise levels, a clear difference can be seen again between the *interleaving* and *alternative participation* configuration and all others, as expected from Figure 72. Between the two outlier configurations one can see that the *alternative participation* configuration performs slightly better for a noise level of 50 %. Yet, this difference does not proove to be statistically significant when testing the difference with the Mann-Whitney test. For the other four configurations, a similar pattern can be observed when comparing these to each other. For noise level 50, the *basic* and *relaxed co–occurrence* configuration do not show any difference to each other in the box plots. Equally, the *relaxed strict order* and *all but interleaving* configuration show an almost identical distribution. While the former group has a lower median of one question, the second group, which uses the relaxed strict order constraints, has a higher median of two questions. Also the 0.75 quartile of the first group is one question lower than that of the second group. In general, such a difference is expected as the relaxation of the strict order constraint increases the search space for those processes that have interleaving behavior in the model. Therefore, more questions need to be asked for those models. Yet, the overall difference between these two groups is not statistically significant. This can be explained by the fact, that the relaxed strict order constraint yields a slightly higher effectiveness that normalizes the effect. For noise levels above 50 % both groups perform almost identically and no difference can be seen in the box plots in Figure 73.

As none of the configurations shows a statistically significant advantage over all others, we will use the *basic* configuration for our further analysis. Figure 74 depicts the number of required questions relative to the number of event classes for the *basic* configuration. The

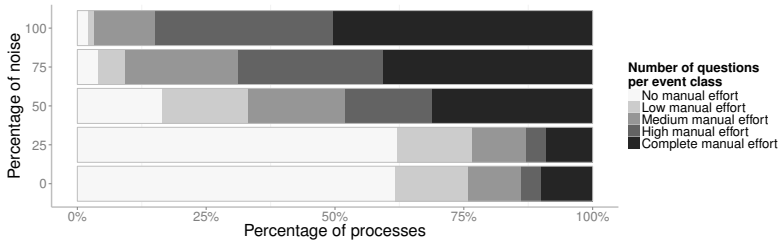


Figure 74: Declare approach – basic configuration: Number of questions per event class for each noise level in one-to-one setting.

share of mappings that could be performed completely automatically is 62 % of all cases with a noise level below 50 %. This is seven percentage points more than for the behavioral profile approach with direct followers and only 3 percentage points less than for the replay approach (65 %). For 14–15 % of all cases with less than 50 % noisy traces the Declare approach required a question for at most every fourth event class (low manual effort). Another 10 % of the cases with a noise level below 50 % could be matched with medium manual effort, i.e., with at most one question for every second event class. Summing this up, 86–87 % of all cases with a noise level below 50 % could be solved with at most medium effort using the Declare approach with its *basic* configuration. In this regard, the results of the Declare approach lie in the middle between the behavioral profile approach with 80-83 % and the replay approach, which handles 90-93 % of the event logs with a noise level below 50 % with at most medium manual effort. Looking at the share of event logs for which a complete manual mapping is required, the Declare approach again performs better than the behavioral profile approach and similar to the replay approach. In that, it leaves 9-10 % of all event logs with less than 50 % noisy traces completely to the analyst.

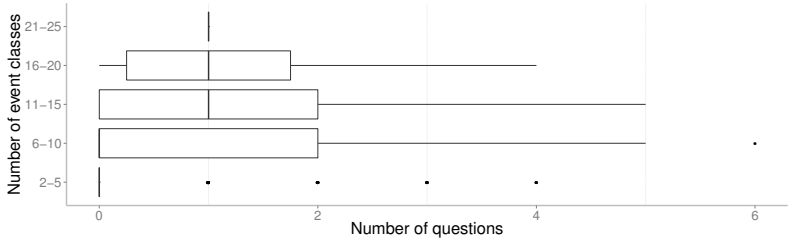


Figure 75: Declare approach – basic configuration: Number of questions per event class for correctly matched event logs without noise in one-to-one setting .

Changing the perspective of the analysis to the number of event classes contained in an event log, Figure 75 shows the previously defined four groups of event class categories and the corresponding box plots for the required number of questions. For event logs with up to five event classes, the Declare approach runs fully automatically for

almost all cases. This is almost identical to the replay approach and thus, far better than the behavioral profile approach, which has to pose up to two questions for a quarter of these event logs. For the category of six to ten event classes, the Declare approach even outperforms the replay approach by handling half of the cases without a question. Only for the category with 16–20 event classes, the replay approach slightly outperforms the Declare approach.

With respect to *(3) the robustness towards noise*, Figures 70–75 already provide insights on how effectiveness and efficiency of the Declare approach change with increasing noise in the event logs. Regarding the effectiveness, Figure 68 revealed that all configurations except for the *interleaving* and *alternative participation* configuration are very robust towards noise. For the two exceptional configurations, we still see a good robustness for the low noise level of 25 % noisy traces. Yet, both rapidly decrease in effectiveness with growing noise, the *alternative participation* configuration a little less than the *interleaving* configuration.

Concerning the efficiency, the Declare approach proves to be stable only until a noise level of 25 %. Beyond this level of noise efficiency drastically drops down. Once more, the Declare approach delivers better results than the behavioral profile approach. For example, the Declare approach is still able to handle 17 % of the event logs where every second trace contains noise completely automatically. This is almost 10 percentage points more than for the behavioral profile approach with direct follower relations. Overall, the Declare approach is still helpful for 70 % of the event logs with 50 % noise and out of these it handles 76 % with at most medium effort. With three quarters of the event logs containing noise, efficiency drops again and gets closer to the results of the behavioral profile approach. Still, even with all traces containing noise, the Declare approach outperforms the behavioral profile approach and thereby also outperforms the replay approach for those cases with noise in every trace.

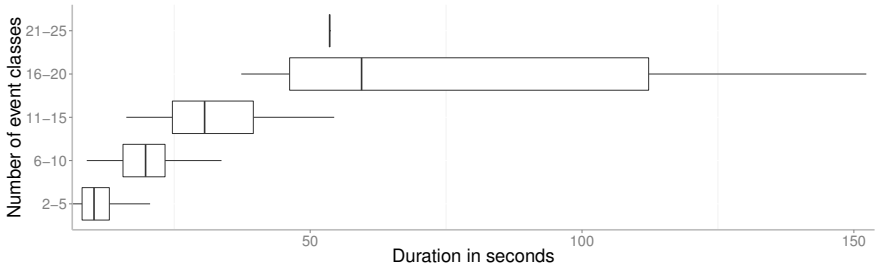


Figure 76: Declare approach – basic configuration: Duration of the matching depending on the number of event classes in one-to-one setting without noise (without outliers).

Turning to *(4) the performance* of the Declare approach, Figure 76 depicts how long the matching takes depending on the number of event classes in the event log. For the group with the fewest event classes, the basic setting of the Declare approach requires less than about ten seconds for half of the matchings. The 0.75 quantile lies at 13 seconds. This is equal to the behavioral profile approach with direct follower relations. Similarly, the durations for the two middle groups are almost equal to those of the behavioral profile approach with direct follower relations. Overall, the Declare approach is slightly faster than the behavioral profile approach, but no significant difference can be found. In the end, these results are far away from those of the replay approach, which also handled the event logs with up to 20 event classes in at most 3 seconds, where both the Declare and the behavioral profile approach take more than 2 minutes.

Figure 77 inspects the performance of the Declare approach with respect to the different noise levels. From this perspective, almost no differences can be found between the Declare approach and the behavioral profile approach. The Declare approach shows slightly better values, yet, there is no significant difference.

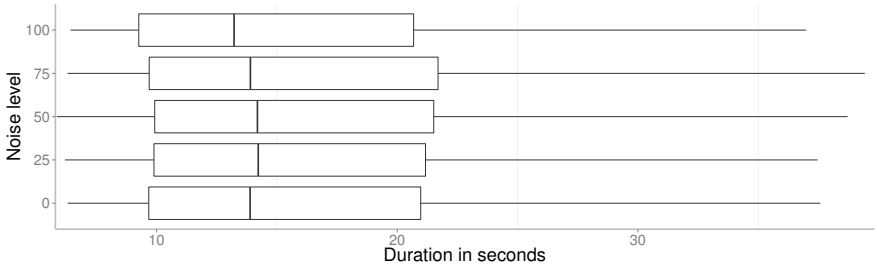


Figure 77: Declare approach – basic configuration: Duration of the matching depending on the noise level in one-to-one setting (without outliers).

Summing up the result for the one-to-one matching, it can be said that the replay approach shows the best performance on all of the four criteria. The Declare approach follows on that and the behavioral profile approach delivered the worst results. Yet, all three approach show very good effectiveness and efficiency for event logs without noise and with little noise. Only the replay approach stays completely stable up to a noise level of 75 %. Nevertheless, it is outperformed by both the Declare and the behavioral profile approach for event logs where every trace contains noise.

8.3.4 *Results for the One-to-many Matching of Activities and Events*

Having evaluated the three approaches that are based on behavior with respect to Requirement R1 (1:1 matching to activities), we now

inspect the fulfillment of Requirement R2 (Different abstraction levels). The replay approach is not able to handle one-to-many relations of activities and events. Therefore, this section only inspects the behavioral profile approach and the Declare approach.

8.3.4.1 *Behavioral Profile Approach*

Starting with the behavioral profile approach, Figure 78 provides the results for the measurement of *(1) the effectiveness*. While the overall pattern looks very similar to the one seen for the one-to-one setting, there is a striking difference in the overall number of event logs that can be matched correctly. For the one-to-many setting the maximum number of correctly matched event logs over all noise levels is 1520, which is 69 % of all 2210 event logs. This is 27 percentage points less than what could be handled in the one-to-one setting. It can be seen that effectiveness slightly decreases for all configurations with increasing noise. Without noise, the most effective configuration is the relaxed strict order configuration, which correctly maps 74 % of all noise-free event logs. Again, over all noise levels, the configuration with direct follower relations scores best and the configuration with interleaving relations scores worst, handling only 42 % of all event logs with decreasing effectiveness on higher noise levels.

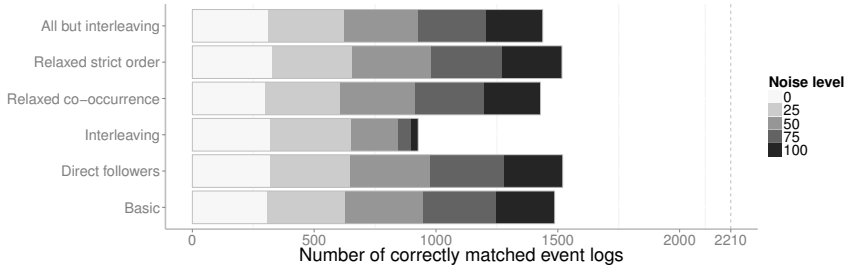


Figure 78: Behavioral profile approach: Number of correctly solved matchings in one-to-many setting for different noise levels.

The root cause analysis for not correctly solved matchings reveals that the majority of those cases fail due to resource shortage. Apart from that, Figure 79 reflects the same problems with order constraints and interleaving constraints as we have seen for the one-to-one setting. Nonetheless, there again is a compelling difference in the order of magnitude. For the one-to-one setting 114 of the matchings failed due to wrong order constraints if these were not relaxed. In the one-to-many setting this number more than doubles, reaching 278 event logs that cannot be matched due to misleading order constraints. This is explained by the increased number of event classes in the event logs which potentiate potential errors. When there are multiple events for each activity of a pair of two interleaving activities, it is more likely that we do not see all ordering combinations of these events equally

often. This potentially leads to a dominant ordering between some of these events, which dominant ordering then results in a conflicting strict order constraint. A similar pattern can be seen for the constraints stemming from interleaving relations. Nonetheless, the effect is not as strong as for the strict order relations. In the one-to-many setting there are only 14 % more conflicting constraints derived from interleaving relations than in the one-to-one setting.

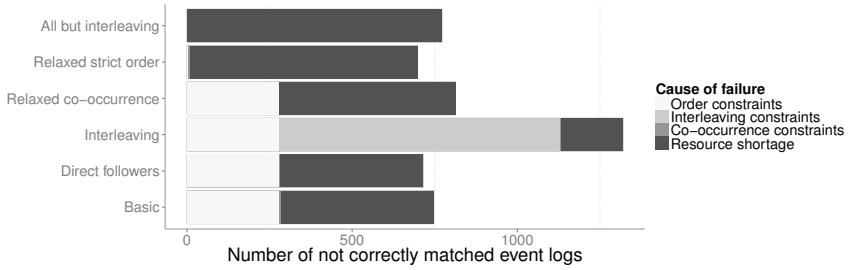


Figure 79: Behavioral profile approach: Number of not correctly matched event logs in the one-to-many setting (all noise levels).

Looking at the developments of conflicting constraints with increasing noise, Figure 80 reveals again the same general patterns as seen in the one-to-one setting. The number of incorrect order constraints decreases with increasing noise while the number of interleaving constraints grows with more noise in the event logs.

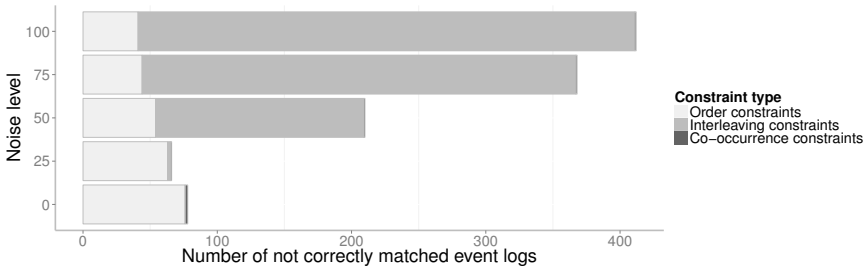


Figure 80: Behavioral profile approach: Number of wrong constraints by type and noise level for the setting with interleaving constraints in the one-to-many setting.

Switching the perspective to the amount of event classes contained in the event logs, Figure 81 lays out how many event logs fall into the different categories and how many of those could be solved or not using the relaxed constraint for strict order relations. While there are now far less event logs with at most five event classes than in the one-to-one setting, these can still all be solved correctly. In the category of six to ten event classes, which is now the largest category, 96 % of all event logs could be solved correctly, which is a little bit less than in the one-to-one setting. The remaining four percent fail due to resource shortage. The share of not solved matchings due to resource shortage rapidly increases to about 25 % for the category of

eleven to fifteen event classes and to 64 % for the next larger category. That is, when there are 16–20 event classes in the event logs, almost two third of the matching cannot be done due to resource shortage. For event logs with more than 20 event classes only one event log can be mapped. Hence, it can clearly be seen that there is a very strong negative relation of the number of event classes and number of mappings that can be correctly solved.

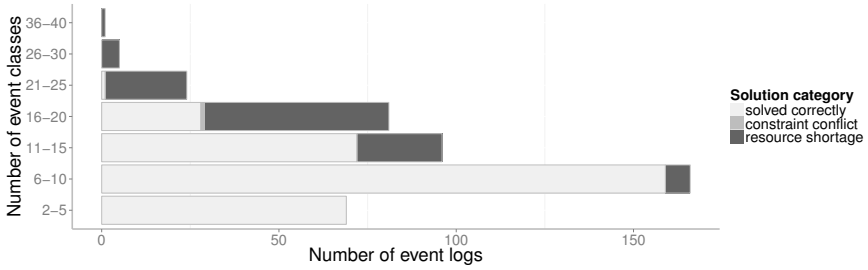


Figure 81: Behavioral profile approach with relaxed strict order constraints: Solution categories by number of event classes for event logs without noise in the one-to-many setting.

With that, we come to *(2) the efficiency* of the behavioral profile approach in the one-to-many setting. Figure 82 provides us with insights on the mean number of questions for each tested configuration for all noise levels. First of all, one can see that there are in general bigger differences between the configurations than in the one-to-one setting. Moreover, one can see that the *interleaving* configuration is no longer a very strong negative outlier. For event logs with no or only little noise, the opposite is true: The *interleaving* configuration performs best for these event logs. Still, for event logs with 75 % or more noisy traces, the *interleaving* configuration needs to ask significantly more questions than all other configurations.

Looking at the ranges in which the average number of questions lies, it can be observed that these are significantly higher than those for the one-to-one setting, which one would expect. For event logs with little or no noise the mean number of questions ranges from 7 to 8.6. This number raises to 8.2 and 9.1 for event logs with 50 % noisy traces and peaks with 10.5 to 11.4 when all event logs contain noise.

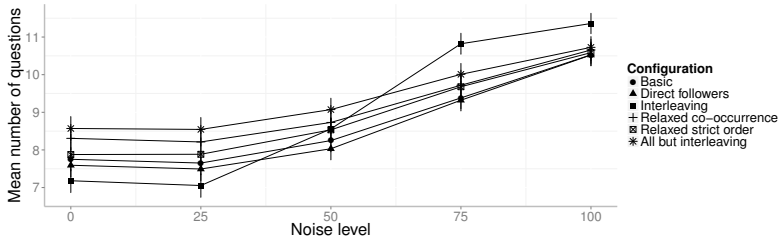


Figure 82: Behavioral profile approach: Mean number of questions for each configuration.

Figure 83 drills down to the distributions of the number of questions by showing the corresponding box plots. It can be seen that the median of the *direct followers* and the *interleaving* configuration is – with five questions – one lower than that of all other configurations for event logs with no or few noise. This is due to the fact that these two configurations impose the strictest constraints onto the mapping. Nonetheless, only for the *interleaving* configuration a statistical significant difference can be proved using the Mann-Whitney test. The difference from the *interleaving* configuration to all others can also be proved statistically significant for noise levels above 50 %, albeit, this time in a negative sense. Calculating the mean number of questions over all noise levels, the *direct followers* configuration scores best. We will therefore continue our more detailed investigation with this configuration.

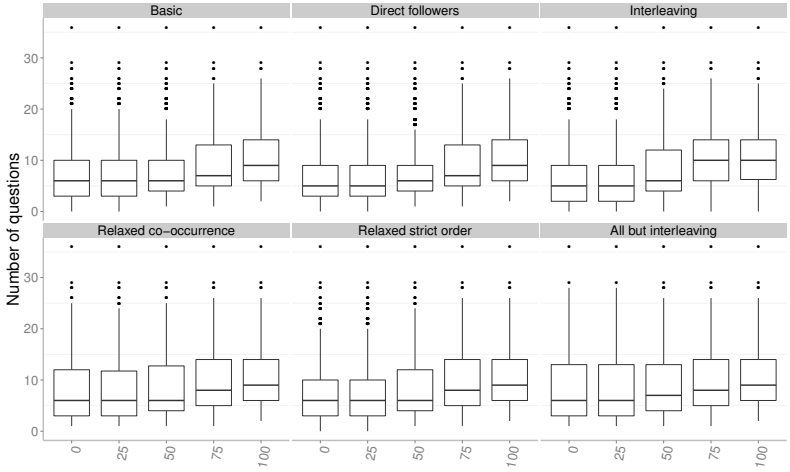


Figure 83: Behavioral profile approach: Boxplots showing the number of required questions for each noise level for each configuration.

Using Figure 84 we analyze the efficiency of the behavioral profile approach by relating the number of questions to the number of event classes, as we did before for the one-to-one setting. What stands out is that there is almost no event log that can be processed completely automatically. Only one event log can be matched completely automatically with no or little noise. With at most medium manual effort 39–40 % of the event logs with noise level zero and 25 can be matched. Again, the small noise level helps in getting rid of incorrect order relations and therefore the approach performs better for these event logs than for logs that are noise-free. Overall, it can be observed that the approach is helpful for 69–72 % of the event logs with no or few noise insertions, which again is significantly less than the 86–87 % that we achieved in the one-to-one setting.

Next, we bring the focus again to the influence of the number of event classes. In contrast to the results for the one-to-one setting, Fig-

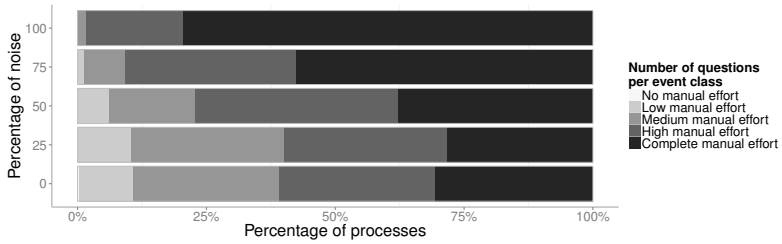


Figure 84: Behavioral profile approach with direct follower relations: Number of questions per event class for each noise level.

ure 85 shows that the number of questions increases within the first three categories and than stagnates or rather decreases for higher categories of event classes. As we only include matchings that were solved correctly, the categories with more than 25 event classes are not shown. For event logs with two to five event classes, the behavioral profile approach with direct follower relations matches 75 % of these logs with at most two questions. For six to ten event classes, at most five questions are required for 75 % of the matchings. Half of the event logs with more than eleven event classes can be matched by asking at most six questions. Summing up, one can again see that the number of questions does not linearly increase with the number of event classes.

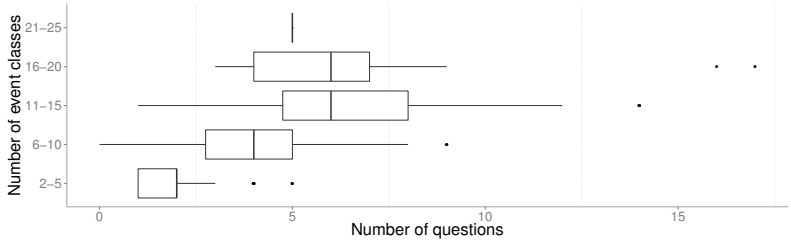


Figure 85: Behavioral profile approach with direct follower relations: Number of questions per event class for correctly matched event logs without noise.

Coming to *(3) the robustness towards noise*, we can again use the insights already provided during the analysis of effectiveness and efficiency. From Figure 84 it can be seen that there is a slight increase in the number of matchings for which the approach is useful when there is a small amount of noise in the event logs compared to when no noise is present. From the noise level of 50 % upwards, this number constantly decreases until there is only a share of 21 % for which the behavioral profile approach helps the analyst. Overall, this development is very similar to that observed in the one-to-one setting, yet, on a much lower order of magnitude and with steeper decrease of effectiveness and efficiency with higher noise levels.

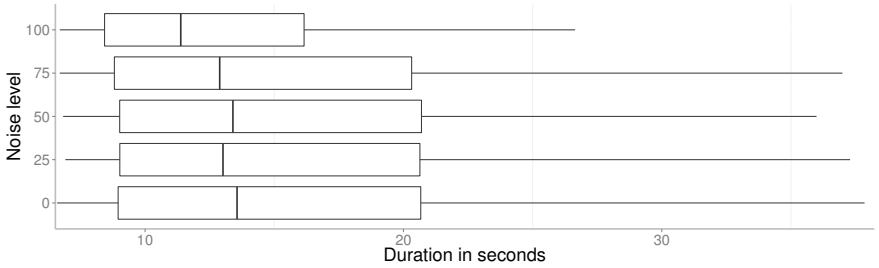


Figure 86: Behavioral profile approach with direct follower relations: Duration of the matching depending on the noise level in the one-to-many setting (without outliers).

For the analysis of *(4) performance*, the durations for solved matchings using the *direct followers* configuration are plotted against the different noise levels in Figure 86. Similarly to the observations made in the one-to-one setting, there is no substantial influence of the amount of noise on the performance of the type–level matching. Also the overall ranges in which the duration lies are very similar to those of the one-to-one setting.

Figure 87 provides the view on the dependency between performance and the number of event classes in the event logs. Once more, a similar pattern can be observed as for the one-to-one matchings. The duration increases exponentially with the number of event classes. Surprisingly, the performance is very similar as in the one-to-one setting and appears to be even slightly better. Yet, this is due to the fact that there are overall less matchings that could be solved. In the end, we still believe that the performance with at most about 2 minutes, is good enough as the type–level matching is a one time undertaking.

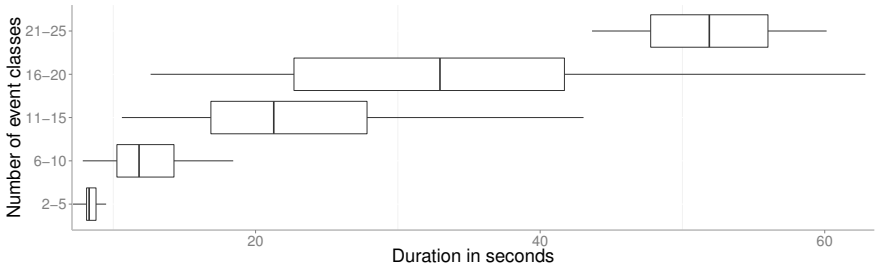


Figure 87: Behavioral profile approach with relaxed mapping for direct follower relations: Duration of the matching depending on the number of event classes in the one-to-many setting (without outliers).

8.3.4.2 *Declare Approach*

Turning to the Declare approach for the matching of activities and events in a one-to-many relation, Figure 88 starts the inspection of *(1) the effectiveness*. First of all, one can observe that the overall effective-

ness ranges on similar levels as for the behavioral profile approach and thus, on far lower levels than in the one-to-one setting. Nevertheless, the Declare approach brings about slightly better numbers than the behavioral profile approach. Yet, this time the two approaches are very close. In the overall effectiveness over all noise levels the Declare approach with its basic configuration reaches 70 % correctly matched event logs, whereas the behavioral profile approach managed to correctly map 69 % with its best configurations. Without any noise the Declare approach reaches 74 % with basic settings and 76 % when employing the relaxed strict order constraints. Here, the maximum effectiveness of the behavioral profile approach is 74 %. Hence, the Declare approach performs slightly better, but the differences are rather small.

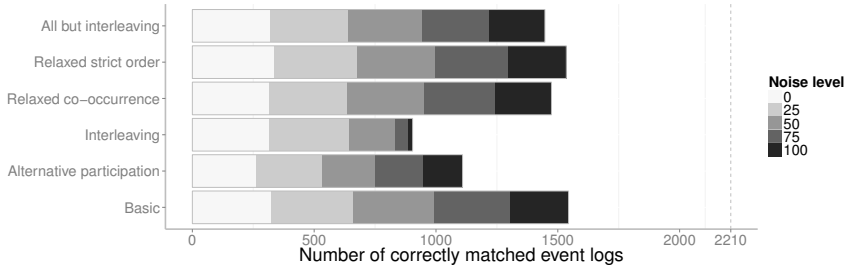


Figure 88: Declare approach: Number of correctly solved matchings in one-to-many setting for different noise levels.

In the analysis of causes for the failed matchings the results look very similar to those of the behavioral profile approach, as depicted in Figure 89. Again, the main cause is resource shortage for most of the configurations. Only for the *alternative participation* and the *interleaving* configuration, the number of conflicting constraints outweighs the resource shortage. Both configurations again show decreasing effectiveness with increasing noise due to the growing number of conflicting constraints.

Since the results for the influence of the number of contained event classes is also almost identical to the behavioral profile approach, we leave out the concrete results. For completeness, Section A in the appendix includes Figure 96, which entails the results. Summing up, it can be seen that the behavioral profile approach and the Declare approach deliver very similar results in terms of effectiveness. Still, the Declare approach performs slightly better.

The *(2) efficiency* of the Declare approach is also almost identical to the behavioral profile approach. For no or only little noise, the *interleaving* configuration proved to be most efficient, as depicted in Figure 90. Nevertheless, when the event logs become more noisy, the *interleaving* configuration again turns out to be worst. The basic configuration delivers the best results for these more noisy event logs and is therefore overall the most efficient configuration. Once more,

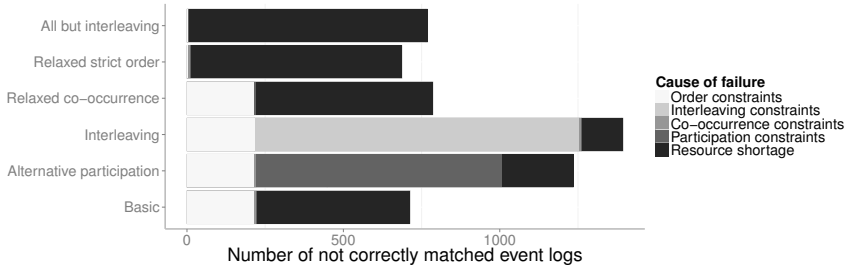


Figure 89: Declare approach: Number of not correctly matched event logs in the one-to-many setting (all noise levels).

there is no significant difference between configurations of the Declare approach and those of the behavioral profile approach and we therefore leave out the further detailed analysis and refer the reader to Section A for the result figures.

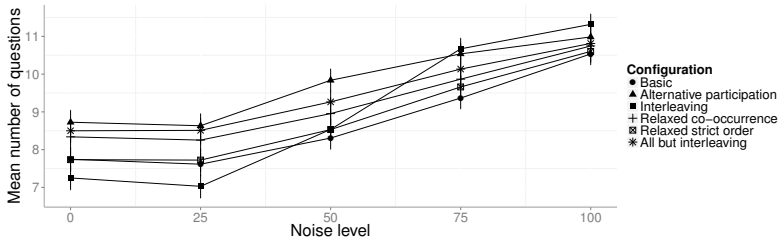


Figure 90: Declare approach: Mean number of questions for each configuration in the one-to-many setting.

With respect to *(3) the robustness towards noise*, the findings for effectiveness and efficiency are repeated. That is, overall, both approaches perform almost identical. Yet, the Declare approach shows a slight advantage over the behavioral profile approach in the overall effectiveness.

For *(4)* the results of the Declare approach are once more almost identical to those of the behavioral profile approach and we will therefore not discuss them further. For completeness, the analysis figures can be found in Section A in the appendix.

8.3.5 *Summary and Discussion*

This section provided an extensive validation of the three different mapping approaches that leverage behavioral knowledge. For the purpose of validation, two sets of event logs have been generated from the BIT process library, which contains hundreds of real life process models from industry. The first set contained event logs for which the events are in a one-to-one relationship to their corresponding activities on type level. For the second set of event logs, event generators were employed in order to simulate activity life cycles that result in a one-to-many relations between activities and events

on type level. Each of the two sets contained five event logs with different noise levels for each process model. For both approaches based on behavioral relations different configurations were tested and compared to each other. In line with the goal of the validation four measures were assessed: (1) effectiveness, (2) efficiency, (3) robustness towards noise, and (4) performance. Table 26 provides a summary of the main results for corresponding measures.

Table 26: Results for behavioral approaches.

	Replay	BP		Declare	
Share of correctly matched event logs (in %)	1:1	1:1	1:N	1:1	1:N
Total	76	94	69	95	70
No noise	93	96	74	97	76
Stable until noise level	75	100	100	100	100
At most medium effort					
No noise	93	80	39	86	38
Stable until noise level	75	25	25	25	25
Completely automatic					
No noise	65	55	0	62	1
Stable until noise level	75	25	100	25	25
Avg. duration (in sec)					
No noise	44	20	23	19	23
Stable until noise level	0	100	100	100	100

Starting with the results for the *one-to-one setting*, the Declare approach with relaxed strict order constraints outperformed the other approaches in (1) effectiveness with 97 % correctly mapped event logs when no noise was contained. The behavioral profile approach with relaxed strict order and the replay approach showed – with effectiveness values of 96 % and 93 % – also very good results for these event logs. For all approaches these values slightly decrease with growing noise in the event logs. While the approaches based on behavioral relations are still able to solve 88–89 % of the CSPs for the matching of event logs where all traces contain noise, the replay approach sees a dramatic decrease in effectiveness for these logs and maps only 17 % correctly due to incorrect constraints. With respect to (3) robustness towards noise, one can say that the approaches based on behavioral relations show rather stable effectiveness values for all noise levels, whereas the replay approach is only robust towards noise until a noise level of 75 % noisy traces in the event logs.

Resource shortage turned out to be the main reason of failure for the top performing configurations of the Declare approach and the behavioral profile approach. The investigation of the other configurations furthermore revealed that strict order relations are most likely to lead to conflicting constraints when the process model contains interleaving activities. Yet, these conflicting constraints decrease with increasing noise because they do not reach the required minimum support. On the contrary, conflicting constraints from interleaving relations and wrong alternative participation constraints heavily increase when more and more traces contain noise. This is due to the fact that with increasing noise more event classes that were previously in an ordering relation or mandatory do not reach the minimum support anymore. Therefore, they are incorrectly seen as interleaving or optional events.

Looking at *(1) the effectiveness* in the *one-to-many setting*, only the approach based on behavioral relations are able to perform the matchings, as a replay is no longer possible. The Declare approach is slightly better as the approach based on behavioral profiles and when it comes to effectiveness in the one-to-many setting. Nevertheless, only 74–76 % of all event logs without noise can be solved and only 69–70 % over all noise levels. Regarding *(3) the robustness towards noise*, again a slight decrease in effectiveness is observed with increasing noise. Still, the results are rather stable and there's no steep decrease for any of the noise levels. The big loss in effectiveness in comparison to the one-to-one setting stems from the increased complexity of the corresponding CSPs, which complexity leads to a high rate of matchings that run out of time or memory. Despite the fact that the results are worse compared to the one-to-one matching, most of the event logs can still be correctly matched.

The *(2) efficiency* has been measured by quantifying the manual work that still has to be done by the analyst in form of questions that need to be answered. In the *one-to-one setting*, the replay approach is significantly more efficient than the other approaches by mapping 93 % of all noise-free event logs with at most medium manual effort. The Declare approach follows with 86 % and the behavioral profile approach comes in last with 80 % correctly matched noise-free event logs with at most medium manual effort. Almost two third of the noise-free event logs are matched completely automatically by the replay approach. Here, the Declare approach comes closer with 62 % and the behavioral profile approach matches 55 % of all noise-free logs without manual interaction. Overall, these values are all very good and show that all the approaches significantly reduce the manual work.

With respect to *(3) robustness towards noise*, the replay approach is also way out in front of the other approaches as the efficiency remains

stable for all noise levels up to the noise level of 75 %. The other approaches see a drastic decrease in efficiency for event logs with 50 % or more noisy traces. The replay approach is ahead of the other approaches due to the fact that most of the noisy traces are simply not considered for the creation of the CSP as they cannot be replayed on the model.

In line with the effectiveness results, the efficiency results for the *one-to-many setting* are orders of magnitude worse than those for the one-to-one setting and both the Declare and the behavioral profile approach perform almost equally. Only 38–39 % of the noise free event logs can be solved with at most medium effort and almost no matching is solved automatically. Only the Declare approach is able to map at least one percent of the noise-free event logs without manual interaction. Nonetheless, the results still show that the approach significantly reduces manual work for almost 40 % of the noise-free event logs.

For *(3) the robustness towards noise*, the results are similar to the one-to-one setting. The efficiency of the approaches based on behavioral relations drops significantly for event logs with 50 % or more noisy traces. Still, this doesn't mean that the approaches are not helpful at all for higher noise levels.

Finally, we measured *(4) the performance* of the approaches. To this end, the Declare approach shows the best values with on average 19–20 seconds for both the one-to-one and the one-to-many setting. The behavioral profile approach is slightly slower, but almost as fast as the Declare approach. The replay approach comes in last by taking more than double the time of the Declare approach for noise-free event logs with a one-to-one relation of their entailed event classes to the activities of the corresponding event log. Overall, the average performance of the approaches seems to be acceptable for the one-time undertaking of the type–level matching.

In the light of our experimental results, the behavioral approaches turned out to be promising, especially with regards to resilience towards moderate noise levels. In a one-to-one setting the approaches require in most of the cases no or only little manual intervention. In one-to-many scenarios where the event logs are on a more fine granular level than the process model, a large share of matchings can be done with moderate manual effort. Still, there are some processes that could not be handled, mainly due to massive parallelism and resulting memory shortage. Future work should investigate how these processes can be handled or, at least, automatically discovered. Potential research directions may look into using heuristics such as relative positions of events and activities. The Triple-S approach introduced in [24], for example, uses relative positions of process model fragments

as part of a means towards process model matching. Such heuristics may be used for the matching to furthermore reduce the search space or even to limit the domains of single variables in the CSP before actually starting to solve the problem. Reducing the search space by minimizing variable domains on the one hand and by introducing further constraints stemming from heuristics on the other hand, could drastically improve performance and allow to handle larger processes. Moreover, further constraints that are not related to control–flow may be derived from event log and model. For example, role information may be used to build constraints that allow mappings only from events produced by a resource that belongs to the same role as indicated for the target activity in the process model. When a process is supported by multiple IT systems and the process model activities are annotated with this information, new constraints can be derived to map only events from specific systems to corresponding activities that are annotated to be executed by these systems. Moreover, decision rules that are annotated to control–flow gateways could be used to create further constraints. Therefore, rules specified, for example, using the Decision Model and Notation (DMN)[11] standard, could be used to detect which path in the process model a certain trace must have taken. Consider the case where a decision table informs whether an order needs to be pre-paid before shipping or can be paid later when the goods are received. In a very simple version this may only depend on the invoice total. When the invoice total is also given as an attribute to the trace or an event in a trace, it can be deducted if an optional activity like "Check prepayment" has been executed. From this knowledge one may furthermore restrict the possible activities that can be matched to the events present in a given trace.

Automatically discovering matchings that cannot be handled to inform the analyst before trying to match, would also be very helpful. To give an example, event logs from processes in which all activities are mandatory and concurrent to each other should not be tried to match with a behavioral approach. The behavioral approaches would not be of any help even if they are able to solve the CSP.

In a similar way, future research needs to analyze how we can predict — based on a given event log and corresponding process model — which of the behavioral approaches and which configuration shall be used for the matching. Possible influence factors for this choice may be the number of interleaving relations and the number of activities in a loop. Potentially, also the size of model and event log in terms of number of activities and number of event classes can be used. Looking at such characteristics of event logs and models and at the outcomes of the different approaches and configurations, a decision tree may be built that informs about when to use which configuration or approach in order to arrive at a correct mapping with minimal

11 See [84] and [33] for more information on DMN.

manual effort.

Finally, research needs to be carried out to investigate how the approach can be extended to support many-to-many relations. That is, cases in which a single event class can be related to multiple activities – e.g. events representing shared functionalities – and single activities are represented by multiple different event classes. In the many-to-many case, the already very large search space for the matching problem grows drastically and other techniques might be necessary to handle this. As a start, event logs containing shared functionalities could be handled with the approach presented in this paper using preprocessing that essentially removes such events. If applicable, the label analysis approach presented in this thesis could be used for the detection of shared functionalities.

8.4 EVALUATION OF THE LABEL ANALYSIS APPROACH

8.4.1 *Evaluation Goals and Setting*

Similar to the behavioral approaches, the label analysis approach builds on the base approach and provides semiautomatic means for the mapping of events and activities on type level. Therefore, this section only focuses on the evaluation of the type–level matching. Again, the goal is to evaluate (1) the *effectiveness* to find the correct result and (2) the *efficiency* in terms of manual work. In contrast to the behavioral approaches, the effectiveness is not a binary choice between finding the right mapping or not finding the right mapping. The label analysis approach may also identify only parts of the mapping. To cope with this fact, we rely on the *recall* metric to measure effectiveness and on the *precision* metric for the efficiency [4]. The recall is calculated by dividing the number of found correct matches by the number of all matches that should have been found. Precision describes the share of correct matches of all identified matches.

Note that an evaluation of the robustness towards nonconforming behavior, which we conducted for the behavioral approaches, is not necessary for the label analysis approach. This is due to the fact that the label analysis approach does not rely on the behavior at all and thus, is not influenced by nonconforming behavior.

For the evaluation of the label analysis approach, we extended the two case studies presented in Section 8.2 by employing the label analysis approach for the type–level matching. For both the incident management and the change management process, detailed work instructions were available and could be used for the matching.

The BIT process library, which we used in the evaluation for the behavioral approaches, is not suited for the evaluation of our label analysis approach presented in Chapter 6. This is due to the fact that

the BIT process library is anonymized and does not contain any labels using natural language.

8.4.2 *Results*

First, we had to link the process model activities with the corresponding work instructions for both processes. The work instructions have been provided in word format in tabular form as it is illustrated in Figure 91. The tables have been converted to CSV files (comma-separated values) and imported into the ProM framework using the dedicated import plug–in described in Section 8.1. Next, the activities in the process model were automatically matched with the work instructions over the given IDs that are also exemplified in Figure 91.

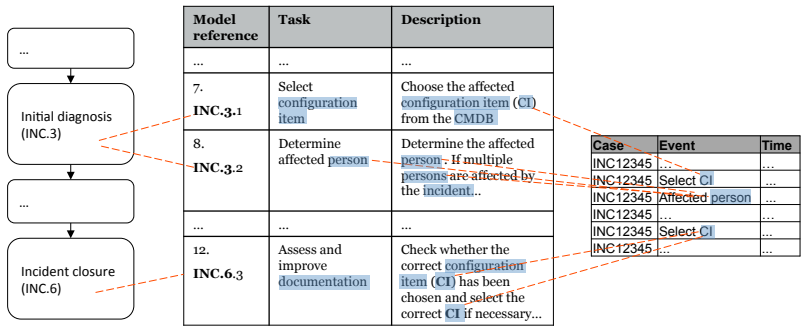


Figure 91: Link between process model activities, work instructions and events.

An overview of the number of annotated activities and used activity descriptions is given in Table 27. Although the process model for the incident process is smaller, a larger amount of textual description with a total of 238 activity descriptions is available. This is due to the documentation of sub–activities that are not part of the process model. The matching algorithm annotated 31 process model activities with 64 descriptions for the incident process. Yet, not all process model activities could be annotated due to missing descriptions. This already provided valuable information for the process manager. For the change process, a smaller set of documenting descriptions was available containing at most one description per activity. Nevertheless, there were only three activities for which there was no description.

Table 27: Process model annotations with activity descriptions.

	Available activity descriptions	Model activities	Annotated activities	Used descriptions
Incident	238	41	31	64
Change	89	63	60	60

Having connected the descriptions to their corresponding activities in the process model, the part-of-speech tagging facilities provided by the Stanford POS tagger was used to automatically extract the potential business objects from the activity descriptions. Once the process model activities were annotated with the potential business object stemming from the work instructions, we turned to the second step of the label analysis approach. In this step, the ProM plug–in also extracted the potential business objects from the event classes and automatically matched events and activities on type level as described in Section 6.4.

In order to assess (1) the effectiveness of the label analysis approach, we measure the *recall* as the number of correctly matched event–activity pairs divided by all manually matched event–activity pairs. From Figure 92 it can be seen that a high recall of 70 % and 86 % is achieved. Figure 93 shows the number of correctly identified event–activity relations. We distinguish between the two types of provenance for a match: the activity name and the description. For both processes, it can be seen that the external knowledge, i.e., the description, accounts for a significantly higher share of correctly identified event–activity relations.

Looking at (2) the efficiency of the label analysis approach, we measure the *precision* as the number of correctly matched event–activity pairs divided by all matched pairs. While the recall is satisfying, Figure 92 shows that precision is in a lower range. For the incident process, the precision of 28.62 % is mainly caused by matches that are based on the additional activity descriptions. Here, we achieve a precision of 26.48 % while the precision of matches on the activity names is high with 64.29 %. The precision for the change process is with 42.58 % substantially higher than the precision for the incident process. Here, the difference between the precision for matches on activity names and matches on the annotated description texts is small. However, description matches account for most of the overall recall, which yields with 70 % and 86.09 % a good result from a practical perspective.

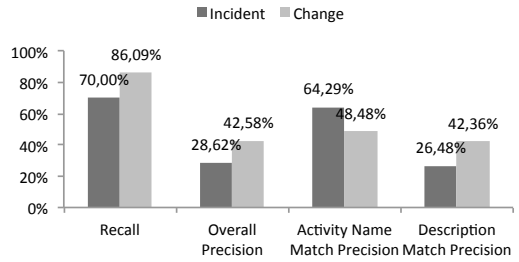


Figure 92: Recall and precision for automated matching.

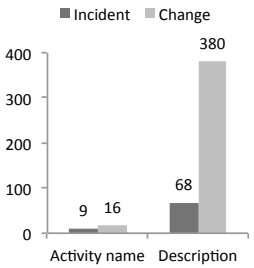


Figure 93: Correct matches by provenance.

We also investigated why certain event–activity relations were not found. It turned out that the main reason is that the sub–activities represented by the events where not documented in the work instructions. Some of these are simply missed out and need to be updated in the description. Here, the approach helped in identifying gaps in the documentation. The other fraction of the undocumented relations are steps that are automatically executed by the system and are therefore missing in the documentation. Future research might investigate in how far such relations can be retrieved from existing software documentations. Beside the undocumented relations, there are two other minor reasons why relations could not be found. First, some relations can only be found by interpreting event attributes. For example there is an event class "Kommunikationsprotokoll" (communication protocol), which contains all events for sent e-mail messages. Looking at the subjects of these messages, which are most often standardized, one could derive further relations. We also encountered one case where the relation could have been established using a verb instead of a business object. However, we did not include verbs in our approach as their inclusion leads to a drastic increase in false positives. Finally, we encountered some mappings that could have been found using synonym relations. However, these synonyms are of a domain-specific nature and not covered in general-purpose tools like WordNet.

8.4.3 *Summary and Discussion*

In this section, the label analysis approach has been evaluated regarding *(1) effectiveness* and *(2) efficiency*. We therefore employed the metrics of *recall* and *precision* on the matching of our two case studies from ITIL processes of a large German outsourcing company. The process models of the change management and the incident management were annotated with descriptions stemming from work instructions.

Concerning *(1) effectiveness*, the label analysis approach yielded a *recall* of 70 % for the incident process and 86 % for the change process. These are satisfying results from a practical perspective since the majority of relations could be derived automatically using label analysis. Our analysis revealed that for both processes the recall heavily profited from the additional descriptions with which we annotated the activities.

The *precision*, with which we measured *(2) efficiency*, turned out to be not as good as the recall. Overall, the label analysis approach resulted in a precision of about 29 % for the incident process and 43 % for the change process. A deeper analysis brought to light that the additional descriptions are mainly responsible for the poor precision results. Looking at the activity labels only, we obtain precision results of 48 % and 64 %. Hence, the additional descriptions proved to be

good and bad at the same time. While the recall significantly profits from these descriptions, the precision is negatively influenced by too many false positives stemming from annotated descriptions.

While the evaluation shows that our approach works good in detecting the relations and gives a satisfying recall for the found event–activity relations, the low precision still leaves the user with a certain amount of wrong relations that need to be sorted out. Here further research is needed to increase precision and to develop methods to make the sorting out more efficient, e.g., by guiding the user in some way. One means of guidance could be the ranking of the results. A ranking could be used to filter out only the most relevant matchings and to present the results in a sorted list where the analyst is likely to find the correct matches within the upper part of the list. Thereby, the analyst may not be required to go through all found relations.

Another way to improve the recall could be to exclude certain words from the matching that potentially lead to many false positives. To this end, metrics such as term frequency–inverse document frequency (tf-idf) may be used. The tf-idf metric sets the frequency of a term in a document in relation to its inverse frequency in all documents [79, p. 177ff]. In our context, a document refers to a label or a description and a term is a business object. Terms with a low tf-idf score are likely to be very common words that should rather not be used for matching. Hence, common words can be identified and excluded from the matching, potentially leading to fewer false positives.

Although the *recall* of the label analysis approach is already quite good, there is still room for improvement. Our analysis revealed that for some events and activities the extraction and use of verbs from the labels and descriptions would improve the recall. Especially in the light of low precision, the use of verbs needs further research. A potential research direction may look into the use of ontologies such as WordNet in order to derive the degree of specialization of a verb. Leopold et al. investigate in [72] different strategies that are partially based on WordNet to derive the granularity of a process model. Similarly, Klinkemüller et al. [67] use label specificity as indicator for the matching of process model elements. Such strategies could also serve as a basis for pruning in the context of the label–based matching. Analyzing the degree of verb specialization with respect to correct matchings and false positives may lead to valuable insights for the employment of verbs in the matching.

Furthermore, general purpose ontologies such as WordNet as well as domain-specific ontologies may be leveraged to use synonyms to further increase the recall. Again, the degree of specialization should be taken into account in the analysis of these additional sources.

8.5 EVALUATION OF THE INTEGRATED APPROACH

8.5.1 *Evaluation Goals and Setting*

In this section, we evaluate the integrated approach for the matching of events and activities, which uses a combination of the behavioral approaches and the label analysis approach. Again, we only focus on the type–level matching as the other parts of the approach have been already evaluated in Section 8.2.

Similar to the course of the evaluation of the behavioral approaches in section Section 8.3, this section evaluates (1) the effectiveness and (2) the efficiency of the integrated approach. For the evaluation of efficiency we will assess both the number of questions required as well as the number of potential mapping activities for an event class. Thereby, we investigate whether the integrated approach provides an advantage over the independent use of a behavioral matching. Finally, we will also report on (3) handling of noise and (4) performance. We refer to the previous section for the definition of these metrics.

Due to the fact that the label analysis approach requires activity and event labels in natural language, the BIT process library cannot be used for this evaluation. The approaches based on behavior on the other hand require a certain minimum conformance and do not support shared functionalities. Therefore, the two real life case studies that have been used for the evaluation of the base approach and the label analysis approach can also not directly be used. In order to enable the integrated approach, we extract a sub-process of the change management process which was known to be very structured and therefore most likely to fulfill the conformance criteria. The extraction both from the process model and event logs could be performed easily. In the process model, the handling of standard changes is split from the general processing very early by using an XOR gateway. Hence, it is straightforward to remove all unnecessary activities from the process model to arrive at a new model that only contains the handling of standard changes. This process model contains seven activities. The filtering of the event log data is also straightforward as standard change cases were already marked by a trace attribute. Removing all traces that do not contain the term "STANDARD CHANGE" in the respective trace attribute from the change process event log, $L_{ChgFull}$, leads to a new event log $L_{StdChgFull}$. This event log contains 649 traces. In order to clean the event log from shared functionalities, the process owner manually identified them. Next the events with shared activities have been either removed from the event log or a simple intermediate mapping has been defined that lifts the identifying event attributes to the class name level. For example for the events of the status event class, the process manager identified a few important status changes that needed to be kept and defined simple mappings that

lift the "value" attribute to the event class level by defining simply mappings. For example, events of the class "Status changed" where the attribute "value" equals to "waiting" have been mapped to a new event class "Status changed waiting" using the mapping techniques of the base approach.

With the goal of meeting the minimum conformance requirements from the behavioral approaches, two more preprocessing steps were performed. First, all incomplete traces were identified and removed. The identification of complete traces could easily be performed by checking if an event of the class "Status changed closed" occurred. Second, we filtered out those variants that included only one case and only kept those with at least two traces. This resulted in a new event log, L_{StdChg}, which contains 364 traces with 5,194 event instances of 14 different event classes. With this event log we performed the type–level matching using the integrated approach and evaluated the aforementioned metrics. The next section will report the results from the matching.

8.5.2 *Results*

The first steps in the integrated approach are formed by the concurrent creation of the two potential activity event class relations. From these two relations, the one created by the selected behavioral approach is used as base relation, while the other one produced by the label analysis approach is used as filter relation.

Starting with the results for *(1) the effectiveness* of the integrated approach, we concentrate on the creation of the base relation because this relation is the critical element. If and only if this relation contains the correct mapping, the approach can effectively solve the matching. We therefore assess which configuration of the behavioral approaches is able to solve the matching correctly. In order to do this, the process manager provided a manual mapping that serves as gold standard against which we check, whether the derived type level mapping is correct or not. If it is not correct, we determine the constraints that are conflicting with the gold standard mapping. As the replay approach does not support one-to-many relations and the event log contains 14 event classes for seven activities in the process model, we can only use the Declare and the behavioral profile approach for the type–level matching to generate the base relation for the integrated approach.

Figures 94 and 95 provide the number of incorrect constraints for each of the previously selected configurations. Two configurations of the behavioral profile approach do not contain any wrong constraints and are therefore able to solve the CSP. All other configurations of the behavioral profile approach fail due to twelve incorrect co–occurrence constraints. The reason for this is that there is one optional activity in the standard change process model for which event instances of one

of the corresponding event classes are almost always present. This event class represents the final measuring of time taken and belongs to an optional quality assurance activity. As this time measurement is performed in almost every case, co–occurrence relations are derived with all event classes for which we also almost always see their event instances. That is, event instances belonging to mandatory activities. Nonetheless, these co–occurrence relations do not exist in the process model since the quality assurance activity is optional and therefore is not part of any co–occurrence relation.

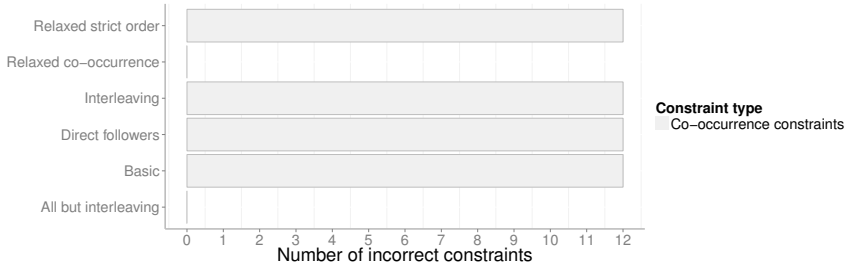


Figure 94: Behavioral profile approach: Number of incorrect constraints by type for the standard change mapping.

For the Declare approach the same incorrect co–occurrence constraints are derived. Additionally, this also leads to an incorrect participation constraint as the time measurement event class is believed to belong to a mandatory activity. While the alternative participation constraint removes this incorrect constraint, it unfortunately produces a different incorrect constraint, albeit, for a different event class. This time a weakness of the alternative participation constraint is revealed. The alternative participation constraint enforces that optional events can only be mapped to optional activities. While this is perfectly fine in a one-to-one setting, it does necessarily work for one-to-many relations between activities and events. In the latter setting there can be optional events for a mandatory activity. Such cases lead to wrong alternative participation constraints. For the standard change there is an optional documentation field that can be filled during the mandatory documentation activity. The event class that reflects the editing of this field leads to the incorrect alternative participation constraint. Thus, none of the configurations of the Declare approach yield a correct mapping.

With the two configurations *relaxed co–occurrence* and *all but interleaving* of the behavioral profile approach, the integrated approach can be successfully applied. We will therefore proceed with these two configurations and turn to the analysis of *(2) the efficiency*. Both configurations lead to the same single question. The one event class for which the analyst needs to decide the mapping activity can poten-

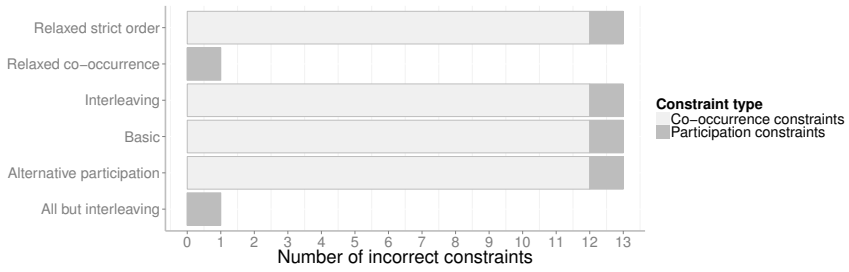


Figure 95: Declare approach: Number of incorrect constraints by type for the standard change mapping.

tially belong to every activity. That is, the user is presented all seven activities of the process model to choose from.

Yet, in the integrated approach, these seven activities are furthermore filtered using the potential activity event class relations derived by the label analysis approach. This relation contains only two matching activities for the event class in question. Due to the very good recall of the label analysis approach, which is 86 % for the standard change process, the correct activity is contained. Therefore, only two activities need to be presented to the user, which is a substantial decrease from the seven activities that the independent behavioral profile approach would have to present.

Finally, we turn to the inspection of *(3) noise* and *(4) performance*. As we are looking at a real life event log from an IT system where the designed process model is not enforced, it is very likely that the event log contains some behavior that is not specified by the process model. In order to inspect the amount of noise contained, we calculated the constraint–relative behavioral profile conformance metric introduced by Weidlich et al. [138]. For the preprocessed event log of the standard change process an overall constraint–relative behavioral profile conformance of 91.87 is achieved. This proves that the filtered event log still contains noise, which is successfully handled by the integrated approach.

Regarding the performance of the integrated approach for type–level matching, we measured the required time until the first question is posed to the user. The behavioral profile approach took in both configurations around 40 seconds to solve the initial CSP. A bit faster, but quite similar, the label analysis approach took about 30 seconds to deliver the potential activity event relations. As both run in parallel, the waiting until the first user interaction has been 40 seconds. We believe that this is still fast enough for the one time undertaking of the type–level matching.

8.5.3 *Summary and Discussion*

This section provided an evaluation of the integrated approach using a real life event log and its corresponding process model from the standard change process. By integrating the approaches based on behavioral relations with the label analysis approach, we furthermore evaluated both the behavioral profile and the Declare approach with real life data.

Looking at *the effectiveness*, it turned out that the behavioral profile approach with relaxed co–occurrence constraints is able to correctly map the event classes of the given event log to the activities of the standard change process model. The Declare approach, in contrast, failed to provide the correct mapping due to a misleading participation constraint.

The *(2) efficiency* of the integrated approach has been tested using the results from the behavioral profile approach and those of the label analysis approach. Only one question is required to perform the mapping. While this question in the behavioral profile approach contains seven different activities to choose from, the number of potential activities can be drastically reduced when using the event–activity relation provided by the label analysis as filtering relation. Finally, the analyst has to choose between only two potential activities.

Again, we could show that the approaches are *(3) robust towards noise*, since the given event log of the standard change log yields only an overall constraint–relative behavioral profile conformance of 91.87. The *(4) performance* of the type–level matching with the integrated approach is measured by looking at the performance of the two integrated approaches individually as both need to complete their calculations before the user interaction can be performed. With a duration of 40 seconds for the slower behavioral profile approach, we believe that the performance is at an acceptable level.

The evaluation of the integrated approach revealed that the participation constraints may lead to conflicting results. Independent from which variant of the participation constraint has been chosen — base or alternative — one of the derived constraint stays in conflict with the correct mapping. Hence, the participation constraint should be made optional in future work. Then the impact of this optionality should be evaluated on a large collection such as the BIT process library.

Moreover, future work should investigate the integration of further techniques to derive event-activity relations on type level, such as the clustering approach derived by Li et al. in [75]. While such a clustering technique may not be sufficient as a base relation, it may be used to derive a filtering relation. Besides investigating the integration of other approaches, future research should also inspect how it is possi-

ble to integrate more than two approaches and whether this proves to be beneficial.

Multiple approaches — and thus, multiple perspectives — can be of great help for the matching of different entities. Klinkmüller et al. [67] introduce an iterative mixed–initiative approach that adapts the matching of process model entities by leveraging the user's feedback. Future work should investigate if and how such an approach may be adopted for the matching of events and activities.

While our evaluation shows promising results for the integrated approach, more case studies are required to fully evaluate the integrated approach on real life data.

8.6 COMPARISON OF MATCHING APPROACHES

Having evaluated each of the different mapping approaches, this section provides a final comparison based on the fulfillment of the nine defined requirements. For those requirements where the approaches differ in their implementation, we will furthermore elaborate on the advantages and disadvantages of the different approaches.

Table 28 entails the comparison of the different introduced matching approaches with respect to the fulfillment of the described requirements from Section 2.4. A fulfilled requirement is marked with a ✓ while a non-fulfilled requirement is indicated by a ×. Requirements that are generally met but impose further assumptions are marked with a •. Overall, it can be seen that only the base approach and the label analysis fulfill all requirements. The other approaches that rely on comparing behavioral aspects are not able to handle shared functionalities, missing events, or additional events. The replay approach is also not able to handle hierarchical mappings since it only supports one-to-one relations between activities and events.

Furthermore, the behavior based approaches make further assumptions for nonconforming event logs. That is, they all request a certain minimal degree of conformance. For the replay approach the majority of traces need to conform to the process model. For the approaches based on behavioral relations, the used behavioral relations found in the event log need to conform to those of the process model with at least a defined minimum support. In our tests and experiments we found 0.9 to be a good support value. Turning the constraint satisfaction problems of the approaches based on behavioral relations into optimization problems, looses this assumption, so that some of the relations in the log may not conform. Yet, our validation and evaluation showed that the optimization problems typically perform worse due to their increased complexity.

Although the behavioral approaches do not fulfill all requirements, they still provide substantial advantages over the base and the label analysis approach when it comes to the required manual effort. Start-

Table 28: Comparison of requirement fulfillment for the different matching approaches.

	Base approach	Replay approach	Beh. rel. approaches	Label analysis approach	Integrated approach
R1 (1:1 matching to activities)	✓	✓	✓	✓	✓
R2 (Different abstraction levels)	✓	×	✓	✓	✓
R3 (Loops and parallelism)	✓	✓	✓	✓	✓
R4 (Shared functionalities)	✓	×	×	✓	×
R5 (Missing events)	✓	×	×	✓	×
R6 (Additional events)	✓	×	×	✓	×
R7 (1:1 matching to life cycle transitions)	✓	✓	✓	✓	✓
R8 (Hierarchical matching)	✓	×	✓	✓	✓
R9 (Nonconforming execution)	✓	•	•	✓	•

ing with Requirement R1 (1:1 matching to activities), we have shown in the evaluation of the behavioral approaches that these are able to significantly reduce the manual work of matching events and activities on type level. For the BIT process library, the replay approach proved to be most efficient by handling more than 90 % of all event logs where at most 75 % of the traces contain noise with at most medium effort. Medium effort means that only for at most every second event class the analyst needs to choose the correct activity. The choice of the correct activity for an event class is furthermore eased by showing only those activities that potentially lead to a correct mapping with respect to the built constraint satisfaction problem. What is more, for those event logs with at most a noise level of 75 %, the replay approach handles almost two third of the matchings completely automatically. Both the Declare approach and the approach based on behavioral profiles cannot compete with the replay approach when it comes to reducing manual work overall. Especially with increasing noise these approaches show drastic decreases in their efficiency. Nonetheless, there are certain processes that cannot be matched by the replay approach, but are solved by the other approaches. Specifically when all traces in the event logs contain noise, the approaches based on behavioral relations outperform the replay approach.

The second requirement, R2 (Different abstraction levels), is not fulfilled by the replay approach. The Declare approach and the approach based on behavioral profiles perform almost equally for event logs that are on a lower abstraction level. Both manage to ease the mapping for almost 40 % of the BIT event logs with no or little noise and require at most medium effort for this share of matchings. In the evaluation with the real life event log from the standard change process, the behavioral profile approach is able to solve the matching and requires only one question. When combined with the label analysis approach, this single question contains the choice between only two activities. The Declare approach fails on this event log due to a conflicting participation constraint.

The label analysis approach also reduces manual work when compared to the base approach. In contrast to the behavioral approaches it is completely robust towards noise since it does not consider any behavioral aspects. Therefore, the label analysis approach could also be applied to the other two real life case studies from the ITIL industry. In these case studies the label analysis approach proved not only to be beneficial for the mapping itself but also pointed the process analysts to gaps in the process documentation.

For all other requirements that have not been discussed so far, all approaches rely on the same techniques as the base approach and a further comparison is not necessary. Summing up, we showed that all of the approaches have their advantages and disadvantages based on which requirements need to be fulfilled. In the one-to-one setting the Replay approach proved to be best in reducing manual work when not all traces contain noise. When all traces contain noise, the Declare approach should be considered. In the one-to-many setting, the behavioral profile approach performed best when event logs contain only little noise, especially, when being integrated with the label analysis approach. For more noisy event logs the label approach should be considered in isolation when natural text labels are available.

9 CONCLUSION

To conclude this thesis, we provide a summary of the results in Section 9.1. Section 9.2 discusses the limitations of our work and outlines research directions for future work.

9.1 SUMMARY OF RESULTS

In this thesis, we introduced novel approaches for the preprocessing of event logs for process mining analysis. Specifically, we tackled the matching problem between activities from a process model and events in an event log which is required especially for conformance and performance analysis but also very beneficial for process discovery. To this end, a novel approach has been introduced to perform such a mapping and different means towards the automation of the mapping have been proposed, validated and evaluated. In particular, the results of this thesis can be summarized as follows.

- *Requirements analysis.* In order to provide a preprocessing for process mining techniques, we first identified nine requirements coming from literature as well as from the experiences in our own case studies. In our literature review on related work, we furthermore demonstrate that none of the currently available mapping approaches is able to fulfill all these requirements.
- *Formalization of the matching between events and activities.* Coming from the requirements analysis, a complete formalization of the mapping problem is provided. To our best knowledge, this is the first work to formalize the mapping of activities and events in the required level of detail concerning all nine requirements.
- *General approach for the matching of events and activities on type and instance level.* Based on the derived formalization, we come up with our base approach for the mapping of activities and events. The base approach is designed to fulfill all nine requirements and operates on two levels: type and instance level. On the type level, flexible concepts are provided to define context–sensitive mappings that take attribute values as well as conditions for the event context into account. These type–level mappings are manually defined and then used to map events on the instance level in an automated fashion. For the mapping on instance level, we furthermore introduce the concept of activity instance borders,

which are used to identify and separate different instances of an activity from each other. These activity instance borders are employed by a tree-based incremental clustering algorithm for the clustering of event instances to activity instances. In addition to the activity instance border definitions, a flexible distance measure can be used for the clustering of activity instances. The base approach has been evaluated in two industry case studies and proved to be a feasible mapping approach with correct results. Our evaluation results furthermore highlight the importance of a correct mapping with respect to conformance and performance results.

- *Automated support of the type–level matching.* Grounded on the base approach, we introduce four innovative techniques to provide automation support for the type–level matching of activities and events. Three of these approaches utilize behavioral aspects of event logs and process models to build a constraint satisfaction problem (CSP) that narrows down potential mappings. Whenever a one-to-one mapping between events and activities is given, our validation showed that the replay approach proved to perform best when it comes to the reduction of manual work. The replay approach handled almost two third of the given event logs with up to 75 % noisy traces completely automatically. In total, about 90 % of these logs were handled with at most medium effort, which demonstrates a great reduction of manual work. Only for very high levels of noise the Declare approach yields better efficiency results than the replay approach. While the replay approach does not handle cases where event log and process model are not on the same abstraction level, the Declare and the behavioral profile approach do, with similar performance for the matchings of the BIT process library. Both approaches manage to create the mappings with at most medium manual effort for almost 40 % of the event logs with no or little noise. For higher noise levels, the approaches still proved beneficial, yet, the manual work increases with increasing noise. The usefulness of the behavioral profile approach has also been demonstrated on a real life event log where 14 event classes could be mapped to seven activities with only one question. Finally, we introduced the label analysis approach, which leverages natural language processing techniques and further external knowledge from textual process documentations for the matching. In contrast to the behavioral approaches, the label analysis approach is independent of the conformance of the event log to the process model. The good performance of the label analysis approach has been demonstrated in two industry case studies.

- *Integrated approach.* Finally, we provide a general approach for the integration of different type–level matching techniques. The integration leverages the results of two type–level matching approaches in order to reduce the amount of potential activities between which an analyst has to choose when a manual decision is necessary for a particular event class. The evaluation on a real life event log revealed that this can lead to drastic improvements. In that case the combination of behavioral profile approach and label analysis led to a reduction from seven to two activities, which is less than a third of the initial items to choose between.

9.2 LIMITATIONS AND FUTURE RESEARCH

While this thesis laid the groundwork for the matching of activities and events, there are also limitations of our work that should be investigated and overcome in future work. This section summarizes the limitations that have been found during our evaluation and gives an outlook on potential research directions for future work.

Looking at the *behavioral approaches*, one of the current drawbacks is that many-to-many relations are not yet supported. Combined with different abstraction levels of event log and process model, shared functionalities lead to a many-to-many relation between events and activities. Due to this, the events of shared functionalities cannot be handled easily. Future work should investigate how behavioral approaches, such as the Declare approach, can be extended to support shared functionalities, i.e., cases in which a single event class can be related to multiple activities. One way of tackling shared functionalities in the behavioral approaches could be another preprocessing step targeting at the discovery of shared functionalities. Once shared activities are discovered, they can be removed in case they are not necessary for the analysis. Otherwise, rules in the form of mappings — as introduced in the base approach — can be used to make distinctions for the events of a shared functionality. With these mappings a preprocessing of the event log can be done before the actual matching is carried out. This preprocessing is what we have already shown for the standard change process in our evaluation in Section 8.5. More research is required on how this could be standardized and supported by automated techniques. Label analysis could potentially be used to detect events that are likely to stem from shared functionalities.

Another way of handling shared functionalities is to restructure the constraint satisfaction problem to support many-to-many relations. While this could technically be done, it imposes different challenges due to the fact that the already very large search space for the matching problem grows drastically when many-to-many relations are possible. First, deriving all solutions to the CSP with reasonable resources

becomes a lot harder. Second, more potential mappings lead to more manual work where a process analyst has to sort out. New techniques and constraints might be necessary to handle this.

Already when activities and events are in a one-to-one or one-to-many relation, some of the event logs in our validation with the BIT process library could not be solved with the given resources. Future research needs to investigate how such cases can be handled. One possibility is to find ways to minimize the search space before the actual solving of the CSP. This can be done by limiting the domains of the variables, i.e., by eliminating potential event–activity combinations upfront. One way of doing this could be to use a positional heuristic that, for example, only allows mappings between activities that appear in the first half of the process model to events that always appear in the first half of the traces. Once more, loops require special attention for such a heuristic.

Moreover, further perspectives could be included in the creation of a CSP. For example by including roles from the organizational perspective. One could use the results of organizational mining or existing knowledge about the roles that are assigned to the users of an IT system to formulate further constraints. Such constraints could, for example, allow only mappings between events and activities that share the same executing role. When multiple IT systems are involved in the execution of a process, the knowledge about which IT system supports which activity and which event stems from which IT system could be used to generate further constraints. Finally, also information on how control–flow routing is done could be integrated to retrieve further constraints. To this end, DMN objects, such as decision tables, could be leveraged to limit the number of activities to which an event class can potentially map.

Regarding the label analysis approach, the precision turned out to be a weak point. To this end, we propose to investigate metrics like tf-idf in order to filter out common terms that lead to a high number of matchings from which potentially many are false positives. Another means towards handling the high number of resulting relations could be the introduction of a ranking metric. Using a ranking, the analyst could be provided only with those relations that are found to be most relevant. Such an ordering could also directly be transferred to the integrated approach.

Our evaluation also showed potential for the improvement of the number of found event–activity relations for the label analysis approach. Here, the use of verbs and synonyms should be investigated in future research. To this end, it is important to also keep a good balance of recall and precision as the increase of recall may come with decrease of precision. One way of approaching this could be to look at the different specialization levels of terms used for the matching

and prioritize those that are more specific.

While this thesis already provides means towards a more automated matching, there are still two aspects left that need to be performed completely manually: the creation of context conditions and the assignment of activity instance border rules. Possibilities to support these tasks in an automated or semi–automated fashion should be developed in future research. Classical methods from the data mining field may be considered for the suggestions of context–sensitive mappings. If an event class is identified as shared functionality, the attribute data of its event instances may be used to apply classical data mining techniques for classification. When the matching activities are known, k-means clustering could for example be applied to separate the event instances into clusters where k equals the number of activities to which the event class refers. Having these clusters, one may derive mapping rules by looking for words or phrases that characterize the instances of a single cluster. These rules can be presented to the analyst for approval or modification.

Part IV

APPENDIX

A

RESULTS OF THE DECLARE APPROACH FOR THE ONE-TO-MANY MATCHING

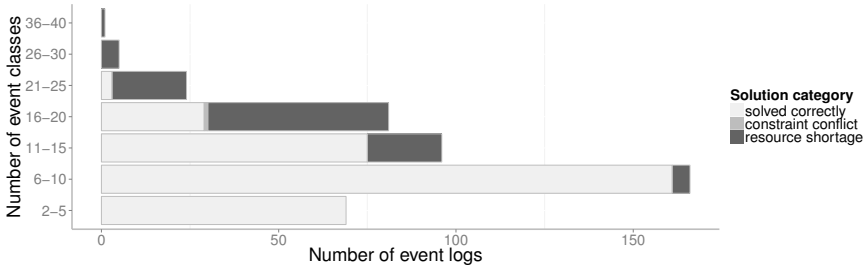


Figure 96: Declare approach with relaxed strict order constraints: Solution categories by number of event classes for event logs without noise in the one-to-many setting.

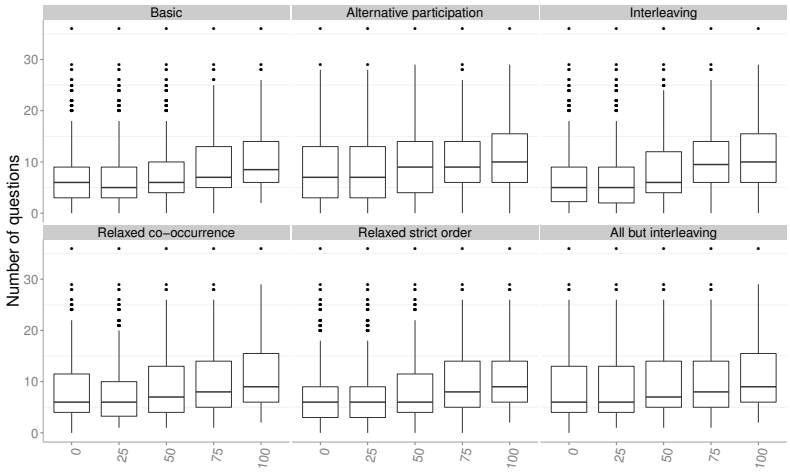


Figure 97: Declare approach: Boxplots showing the number of required questions for each noise level for each configuration in the one-to-many setting.

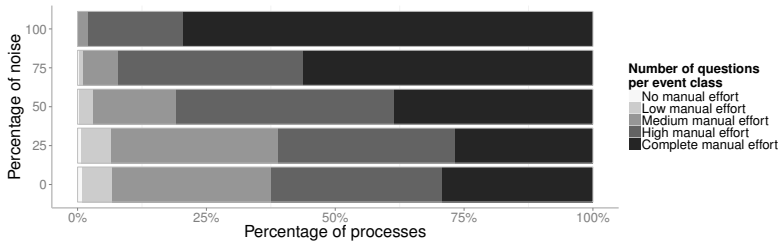


Figure 98: Declare approach - basic configuration: Number of questions per event class for each noise level in the one-to-many setting.

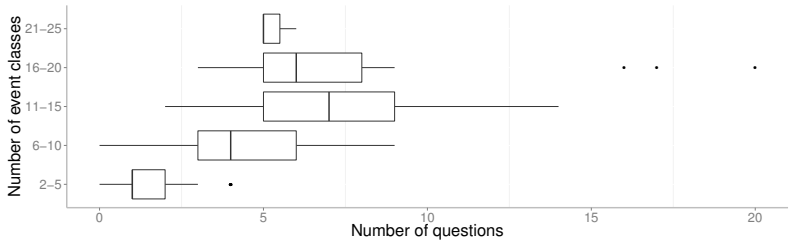


Figure 99: Declare approach - basic configuration: Number of questions per event class for event logs without noise in the one-to-many setting.

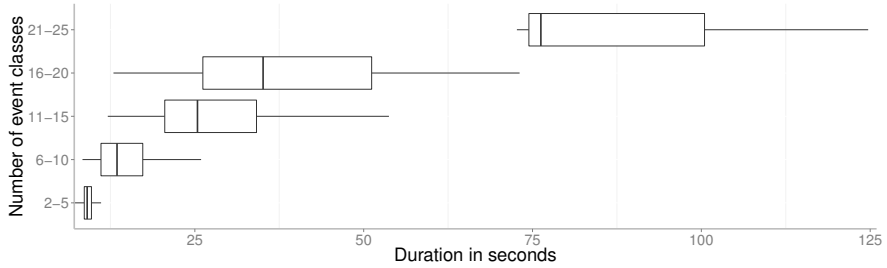


Figure 100: Declare approach - basic configuration: Duration of the one-to-many matching depending on the noise level (without outliers).

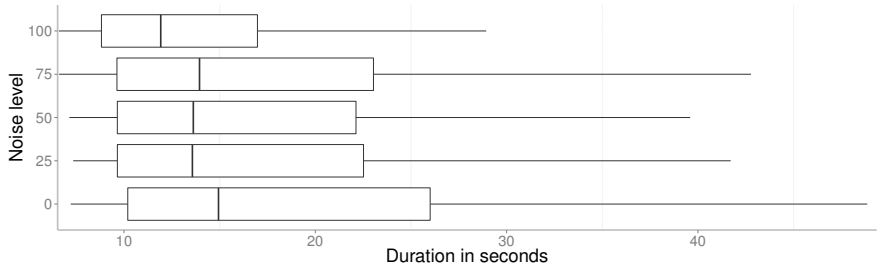


Figure 101: Declare approach - basic configuration: Duration of the matching depending on the number of event classes (without outliers).

BIBLIOGRAPHY

[1] Sven Abels and Axel Hahn. Pre-processing Text for Web Information Retrieval Purposes by Splitting Compounds into their Morphemes. In *OSWIR*, 2005.

[2] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003. Cambridge Books Online.

[3] Abel Armas-Cervantes, Marlon Dumas, Luciano García-Bañuelos, and Artem Polyvyanyy. On the suitability of generalized behavioral profiles for process model comparison. In *11th International Workshop on Web Services and Formal Methods (WS-FM)*, 2014.

[4] R. A. Baeza-Yates and B.A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[5] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[6] Thomas Baier and Jan Mendling. Bridging abstraction layers in process mining by automated matching of events and activities. In *Business Process Management - 11th International Conference*, volume 8094 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2013.

[7] Thomas Baier and Jan Mendling. Bridging abstraction layers in process mining: Event to activity mapping. In *Enterprise, Business-Process and Information Systems Modeling - 14th International Conference*, volume 147 of *Lecture Notes in Business Information Processing*, pages 109–123. Springer, 2013.

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

[9] Thomas Baier, Andreas Rogge-Solti, Mathias Weske, and Jan Mendling. Matching of events and activities - an approach based on constraint satisfaction. In *The Practice of Enterprise Modeling - 7th IFIP WG 8.1 Working Conference*, volume 197 of *Lecture Notes in Business Information Processing*, pages 58–72. Springer, 2014.

[10] Thomas Baier, Claudio Di Ciccio, Jan Mendling, and Mathias Weske. Matching of events and activities - an approach using declarative modeling constraints. In *Enterprise, Business-Process*

and Information Systems Modeling - 16th International Conference, volume 214 of *Lecture Notes in Business Information Processing*, pages 119–134. Springer, 2015.

[11] Thomas Baier, Andreas Rogge-Solti, Mathias Weske, and Jan Mendling. Matching of events and activities - an approach based on behavioral constraint satisfaction. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1225–1230. ACM, 2015.

[12] Seyed-Mehdi-Reza Beheshti, Boualem Benatallah, Hamid R. Motahari Nezhad, and Sherif Sakr. A query language for analyzing business processes execution. In *BPM*, pages 281–297, 2011.

[13] Eike Best and Harro Wimmel. Structure theory of petri nets. *T. Petri Nets and Other Models of Concurrency*, 7:162–224, 2013.

[14] Egon Börger and Bernhard Thalheim. Modeling workflows, interaction patterns, web services and business processes: The asm-based approach. In *Abstract State Machines, B and Z, First International Conference, ABZ 2008, London, UK, September 16-18, 2008. Proceedings*, pages 24–38, 2008.

[15] R. P. Jagadeesh Chandra Bose and W. M. P. van der Aalst. Abstractions in process mining: A taxonomy of patterns. In *BPM'2009, volume 5701 of LNCS*, pages 159–175. Springer, 2009.

[16] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Process diagnostics using trace alignment: Opportunities, issues, and challenges. *Inf. Syst.*, 37(2):117–141, 2012.

[17] R. P. Jagadeesh Chandra Bose, H. M. W. (Eric) Verbeek, and Wil M. P. van der Aalst. Discovering hierarchical process models using prom. volume 107 of *LNBIP*, pages 33–48. Springer, 2011.

[18] R. P. Jagadeesh Chandra Bose, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. Enhancing declare maps based on event correlations. In *BPM*, pages 97–112, 2013.

[19] Melike Bozkaya, Joost Gabriels, and Jan Martijn E. M. van der Werf. Process diagnostics: A method based on process mining. In *International Conference on Information, Process, and Knowledge Management, eKNOW 2009, Cancun, Mexico, February 1-7, 2009*, pages 22–27, 2009.

[20] Moisés Castelo Branco, Javier Troya, Krzysztof Czarnecki, Jochen Malte Küster, and Hagen Völzer. Matching business process workflows across abstraction levels. In *MoDELS*, pages 626–641, 2012.

[21] Martin Braschler and Bärbel Ripplinger. How Effective is Stemming and Decompounding for German Text Retrieval? *IR*, 7 (3/4):291–316, September 2004.

[22] Carmen Bratosin, Natalia Sidorova, and Wil M. P. van der Aalst. Distributed genetic process mining using sampling. In *Parallel Computing Technologies - 11th International Conference, PaCT 2011, Kazan, Russia, September 19-23, 2011. Proceedings*, pages 224–237, 2011.

[23] David Cannon and David Wheeldon. *ITIL – Service Operation*. TSO, May 2007.

[24] Ugur Cayoglu, Remco M. Dijkman, Marlon Dumas, Peter Fettke, Luciano García-Bañuelos, Philip Hake, Christopher Klinkmüller, Henrik Leopold, André Ludwig, Peter Loos, Jan Mendling, Andreas Oberweis, Andreas Schoknecht, Eitam Sheetrit, Tom Thaler, Meike Ullrich, Ingo Weber, and Matthias Weidlich. Report: The process model matching contest 2013. In *Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, pages 442–463, 2013.

[25] Claudio Di Ciccio, Massimo Mecella, and Jan Mendling. The effect of noise on mined declarative constraints. *unknown???*, 2015.

[26] Diane J. Cook, Narayanan Chatapuram Krishnan, and Parisa Rashidi. Activity discovery and activity recognition: A new partnership. *IEEE T. Cybernetics*, 43(3):820–828, 2013.

[27] Ajantha Dahanayake, Richard J. Welke, and Gabriel Cavalheiro. Improving the understanding of bam technology for real–time decision support. *Int. J. Bus. Inf. Syst.*, 7(1):1–26, December 2011.

[28] T.H. Davenport. *Process Innovation: Reengineering Work Through Information Technology*. Harvard Business Review Press, 2013.

[29] Rob Davis and Eric Brabander. *ARIS Design Platform - Getting started with BPM*. Springer, Berlin, 2007.

[30] H. de Beer. The LTL Checker Plugins: A Reference Manual. Eindhoven University of Technology, Eindhoven, 2004.

[31] A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters. Process mining: Extending the α-algorithm to mine short loops. In *Eindhoven University of Technology, Eindhoven*, 2004.

[32] Ana Karla A. de Medeiros, A. J. M. M. Weijters, and Wil M. P. van der Aalst. Genetic process mining: an experimental evaluation. *Data Min. Knowl. Discov.*, 14(2):245–304, 2007.

[33] Tom Debevoise and James Taylor. *The MicroGuide to Process Modeling and Decision in BPMN/DMN*. CreateSpace Independent Publishing Platform, 2014.

[34] Gero Decker and Jan Mendling. Process instantiation. *Data Knowl. Eng.*, 68(9):777–792, 2009.

[35] Juliane Dehnert and Peter Rittgen. Relaxed soundness of business processes. In KlausR. Dittrich, Andreas Geppert, and MoiraC. Norrie, editors, *Advanced Information Systems Engineering*, volume 2068 of *Lecture Notes in Computer Science*, pages 157–170. Springer Berlin Heidelberg, 2001.

[36] Claudio Di Ciccio and Massimo Mecella. A two-step fast algorithm for the automated discovery of declarative workflows. In *CIDM*, pages 135–142. IEEE, 2013.

[37] Claudio Di Ciccio and Massimo Mecella. Mining artful processes from knowledge workers' emails. *IEEE Internet Computing*, 17(5):10–20, 2013.

[38] Claudio Di Ciccio and Massimo Mecella. On the discovery of declarative control flows for artful processes. *ACM Trans. Manage. Inf. Syst.*, 5(4):24:1–24:37, 2015.

[39] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling. Similarity of Business Process Models: Metrics and Evaluation. *Information Systems*, 36(2):498–516, 2011.

[40] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Information & Software Technology*, 50(12):1281–1294, 2008.

[41] Remco M. Dijkman, Marlon Dumas, Luciano García-Bañuelos, and Reina Käärik. Aligning business process models. In *Proceedings of the 13th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2009, 1-4 September 2009, Auckland, New Zealand*, pages 45–53, 2009.

[42] Marlon Dumas, Wil M. P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems*. John Wiley & Sons, Inc., 2005.

[43] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013.

[44] Robert Engel, Wil M. P. van der Aalst, Marco Zapletal, Christian Pichler, and Hannes Werthner. Mining inter-organizational business process models from EDI messages: A case study from

the automotive sector. In *Advanced Information Systems Engineering - 24th International Conference, CAiSE 2012, Gdansk, Poland, June 25-29, 2012. Proceedings*, pages 222–237, 2012.

[45] Opher Etzion and Peter Niblett. *Event Processing in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2010.

[46] J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer-Verlag, 2007.

[47] Dirk Fahland and Wil M. P. van der Aalst. Simplifying discovered process models in a controlled manner. *Inf. Syst.*, 38(4): 585–605, 2013.

[48] Dirk Fahland and Wil M. P. van der Aalst. Model repair - aligning process models to reality. *Inf. Syst.*, 47:220–243, 2015.

[49] Dirk Fahland, Daniel Lübke, Jan Mendling, Hajo A. Reijers, Barbara Weber, Matthias Weidlich, and Stefan Zugal. Declarative versus imperative process modeling languages: The issue of understandability. In *Enterprise, Business-Process and Information Systems Modeling, 10th International Workshop, BPMDS 2009, and 14th International Conference, EMMSAD 2009, held at CAiSE 2009, Amsterdam, The Netherlands, June 8-9, 2009. Proceedings*, pages 353–366. 2009.

[50] Dirk Fahland, Jan Mendling, Hajo Reijers, Barbara Weber, Matthias Weidlich, and Stefan Zugal. Declarative versus imperative process modeling languages: The issue of maintainability. In *Business Process Management Workshops*, pages 477–488, 2009.

[51] Dirk Fahland, Cédric Favre, Jana Koehler, Niels Lohmann, Hagen Völzer, and Karsten Wolf. Analysis on demand: Instantaneous soundness checking of industrial business process models. *Data & Knowledge Engineering*, 70(5):448–466, 2011.

[52] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Mining predictive process models out of low-level multidimensional logs. In *Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings*, pages 533–547, 2014.

[53] Eugene Freuder and Alan Mackworth. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, chapter Constraint satisfaction: An emerging paradigm, pages 13–27. Elsevier, 2006.

[54] Kerstin Gerke, Jorge Cardoso, and Alexander Claus. Measuring the compliance of processes with reference models. volume 5870 of *LNCS*, pages 76–93. Springer, 2009.

[55] Gianluigi Greco, Antonella Guzzo, and Luigi Pontieri. Mining taxonomies of process models. *Data & Knowledge Engineering*, 67(1):74–102, October 2008.

[56] The Object Management Group. Business Process Modeling Notation (BPMN), version 2.0. available at: http://www.omg.org/spec/BPMN/2.0/PDF, March 2011. Version 2.0,.

[57] Christian W. Günther and Wil M. P. van der Aalst. Mining activity clusters from low-level event logs. In *BETA Working Paper Series*, volume WP 165. Eindhoven University of Technology, 2006.

[58] Christian W. Günther and Wil M. P. van der Aalst. Fuzzy mining: adaptive process simplification based on multi-perspective metrics. In *BPM'2007*, pages 328–343. Springer, 2007.

[59] Christian W. Günther, Anne Rozinat, and Wil M. P. van der Aalst. Activity mining by global trace segmentation. In *BPM Workshops*, pages 128–139, 2009.

[60] M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. Harper Business, 1993.

[61] Michael Hammer. What is business process management? In Janvom Brocke and Michael Rosemann, editors, *Handbook on Business Process Management 1*, International Handbooks on Information Systems, pages 3–16. Springer Berlin Heidelberg, 2010.

[62] Nico Herzberg, Andreas Meyer, and Mathias Weske. An Event Processing Platform for Business Process Management. In *Enterprise Distributed Object Computing Conference (EDOC)*, pages 107–116, Vancouver, 2013. IEEE.

[63] D. Jurafsky and J.H. Martin. *Speech and language processing*. Prentice Hall, 2008.

[64] Narendra Jussien, Guillaume Rochart, and Xavier Lorca. Choco: an open source java constraint programming library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pages 1–10, 2008.

[65] Kimmo Kettunen, Markus Sadeniemi, Tiina Lindh-Knuutila, and Timo Honkela. Analysis of eu languages through text compression. In *FinTAL*, volume 4139 of *LNCS*, pages 99–109. Springer, 2006.

[66] Christopher Klinkmüller, Ingo Weber, Jan Mendling, Henrik Leopold, and André Ludwig. Increasing recall of process model

matching by improved activity label matching. In *BPM*, pages 211–218, 2013.

[67] Christopher Klinkmüller, Henrik Leopold, Ingo Weber, Jan Mendling, and André Ludwig. Listen to me: Improving process model matching through user feedback. In *Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings*, pages 84–100, 2014.

[68] A. Knopfel, B. Grone, and P. Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. Timely. practical. reliable. Wiley, 2005.

[69] Matthias Kunze, Matthias Weidlich, and Mathias Weske. Behavioral similarity - a proper metric. In *BPM*, pages 166–181, 2011.

[70] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from incomplete event logs. In *Application and Theory of Petri Nets and Concurrency - 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Proceedings*, pages 91–110, 2014.

[71] H. Leopold, M. Niepert, M. Weidlich, J. Mendling, R. Dijkman, and H. Stuckenschmidt. Probabilistic optimization of semantic process model matching. In *BPM'2012*, pages 319–334, 2012.

[72] Henrik Leopold, Fabian Pittke, and Jan Mendling. Towards measuring process model granularity via natural language analysis. In *Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, pages 417–429, 2013.

[73] Henrik Leopold, Jan Mendling, Hajo A. Reijers, and Marcello La Rosa. Simplifying process model abstraction: Techniques for generating model names. *Inf. Syst.*, 39:134–151, 2014.

[74] Jiafei Li, Dayou Liu, and Bo Yang. Process mining: Extending α-algorithm to mine duplicate tasks in process logs. In KevinChen-Chuan Chang, Wei Wang, Lei Chen, ClarenceA. Ellis, Ching-Hsien Hsu, AhChung Tsoi, and Haixun Wang, editors, *Advances in Web and Network Technologies, and Information Management*, volume 4537 of *Lecture Notes in Computer Science*, pages 396–407. Springer Berlin Heidelberg, 2007.

[75] Jiafei Li, RPJC Bose, and W. M. P. van der Aalst. Mining context-dependent and interactive business process maps using execution patterns. In *BPM'2010 Workshops, volume 66 of LNBIP*, pages 109–121. Springer, 2011.

[76] Jiexun Li, HarryJiannan Wang, and Xue Bai. An intelligent approach to data extraction and task identification for process mining. *Information Systems Frontiers*, pages 1–14, 2015.

[77] Niels Lohmann, Eric Verbeek, and Remco M. Dijkman. Petri net transformations for business processes - A survey. *T. Petri Nets and Other Models of Concurrency*, 2:46–63, 2009.

[78] Fabrizio Maria Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In *CAiSE*, pages 270–285, 2012.

[79] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[80] Ronny S Mans, MH Schonenberg, Minseok Song, Wil M. P. van der Aalst, and Piet JM Bakker. *Application of process mining in healthcare–a case study in a dutch hospital*. Springer, 2009.

[81] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[82] Hamid R. Motahari Nezhad, Régis Saint-Paul, Fabio Casati, and Boualem Benatallah. Event correlation for process discovery from web service interaction logs. *VLDB J.*, 20(3):417–444, 2011.

[83] Object Management Group (OMG). OMG Unified Modeling Language (OMG UML): Superstructure, version 2.4.1. Available at: `http://www.omg.org/spec/UML/2.4.1/`, August 2011. Version 2.2, formal/2009-02-02, The Object Management Group.

[84] Object Management Group (OMG). Decision Model and Notation, November 2014.

[85] Ricardo Pérez-Castillo, Barbara Weber, Jakob Pinggera, Stefan Zugal, Ignacio García Rodríguez de Guzmán, and Mario Piattini. Generating event logs from non-process-aware systems enabling business process mining. *Enterprise IS*, 5(3):301–335, 2011.

[86] Ricardo Pérez-Castillo, Barbara Weber, Ignacio García Rodríguez de Guzmán, Mario Piattini, and Jakob Pinggera. Assessing event correlation in non-process-aware information systems. *Software and System Modeling*, 13(3):1117–1139, 2014.

[87] Maja Pesic. *Constraint-Based Workfow Management Systems: Shifting Control to Users*. PhD thesis, Technische Universiteit Eindhoven, 2008.

[88] Maja Pesic, M.H. Schonenberg, Natalia Sidorova, and Wil M. P. van der Aalst. Constraint-based workflow models: Change made easy. In *OTM Conferences (1)*, pages 77–94, 2007.

[89] C.A. Petri. Fundamentals of a Theory of Asynchronous Information Flow. In *Proceedings of the Information Processing Congress (IFIP Congress)*, August/September 1962.

[90] Paul Pichler, Barbara Weber, Stefan Zugal, Jakob Pinggera, Jan Mendling, and Hajo A. Reijers. Imperative versus declarative process modeling languages: An empirical investigation. In *Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I*, pages 383–394, 2011.

[91] Amir Pnueli. The Temporal Logic of Programs. In *Foundations of Computer Science*, pages 46–57, 1977.

[92] Artem Polyvyanyy, Sergey Smirnov, and Mathias Weske. Process Model Abstraction: A Slider Approach. In *EDOC*, pages 325–331. IEEE, 2008.

[93] Artem Polyvyanyy, Sergey Smirnov, and Mathias Weske. On application of structural decomposition for process model abstraction. In *BPSC*, pages 110–122, 2009.

[94] Artem Polyvyanyy, Matthias Weidlich, and Mathias Weske. Isotactics as a foundation for alignment and abstraction of behavioral models. In *BPM*, pages 335–351, 2012.

[95] Elham Ramezani, Dirk Fahland, and WilM.P. van der Aalst. Where did i misbehave? diagnostic information in compliance checking. In Alistair Barros, Avigdor Gal, and Ekkart Kindler, editors, *Business Process Management*, volume 7481 of *Lecture Notes in Computer Science*, pages 262–278. Springer Berlin Heidelberg, 2012.

[96] Hajo A. Reijers, Tijs Slaats, and Christian Stahl. Declarative modeling: An academic dream or the future for bpm? In *Proceedings of the 11th International Conference on Business Process Management*, BPM'13, pages 307–322, Berlin, Heidelberg, 2013. Springer-Verlag.

[97] Wolfgang Reisig. *Petri Nets: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[98] Andreas Rogge-Solti and Mathias Weske. Prediction of remaining service execution time using stochastic Petri nets with arbitrary firing delays. In *Service-Oriented Computing*, volume 8274 of *LNCS*, pages 389–403. Springer Berlin Heidelberg, 2013.

[99] Marcello La Rosa, Johannes W. Lux, Stefan Seidel, Marlon Dumas, and Arthur H.M. ter Hofstede. Questionnaire-driven configuration of reference process models. *LNCS*, pages 424–438, 2006.

[100] A. Rozinat and Wil M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64 – 95, 2008.

[101] Anne Rozinat and Wil M. P. van der Aalst. Decision mining in prom. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 420–425. Springer, 2006.

[102] Anne Rozinat, Ivo S. M. de Jong, Christian W. Günther, and Wil M. P. van der Aalst. Process mining applied to the test process of wafer scanners in ASML. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 39(4):474–479, 2009.

[103] Szabolcs Rozsnyai, Aleksander Slominski, and Geetika T. Lakshmanan. Discovering event correlation rules for semi-structured business processes. In *DEBS*, pages 75–86, 2011.

[104] Scheer, August-Wilhelm. *ARIS - Modellierungsmethoden, Metamodelle, Anwendungen, 4th edition*. Springer-Verlag, 2001.

[105] Robert M. Shapiro, Stephen A. White, Conrad Bock, Nathaniel Palmer, Michael zur Muehlen, Marco Brambilla, and Denis Gagné et al. *BPM 2.0 Handbook – Methods, Concepts, Case Studies and Standards in Business Process Modeling Notation (BPMN)*. Future Strategies Inc., 2nd edition edition, 2012.

[106] Sergey Smirnov, Hajo A. Reijers, and Mathias Weske. From fine-grained to abstract process models: A semantic approach. *Inf. Syst.*, 37(8):784–797, 2012.

[107] Sergey Smirnov, Matthias Weidlich, and Jan Mendling. Business process model abstraction based on synthesis from well-structured behavioral profiles. *Int. J. Cooperative Inf. Syst.*, 21(1): 55–83, 2012.

[108] H. Smith and P. Fingar. *Business Process Management: The Third Wave*. Meghan-Kiffer Press, 2007.

[109] H. Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, 1973.

[110] Mirko Steinle, Karl Aberer, Sarunas Girdzijauskas, and Christian Lovis. Mapping moving landscapes by mining mountains of logs: Novel techniques for dependency model generation. In *VLDB*, pages 1093–1102, 2006.

[111] P. S. Thiagarajan and Klaus Voss. In praise of free choice nets. In *Advances in Petri Nets 1984, European Workshop on Applications and Theory in Petri Nets, covers the last two years which include the workshop 1983 in Toulouse and the workshop 1984 in Aarhus, selected papers*, pages 438–454, 1984.

[112] K. Toutanova and Ch. D. Manning. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. *EMNLP*, pages 63–70, 2000.

[113] W. M. P. van der Aalst and A. J. M. M. Weijters. Process mining: a research agenda. *Computers in Industry*, 53, April 2004.

[114] Wil M. P. van der Aalst. Verification of workflow nets. In *ICATPN*, volume 1248 of *LNCS*, pages 407–426. Springer, 1997.

[115] Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8 (1):21–66, 1998.

[116] Wil M. P. van der Aalst. Process-aware information systems: Lessons to be learned from process mining. *T. Petri Nets and Other Models of Concurrency*, 2:1–26, 2009.

[117] Wil M. P. van der Aalst. Process mining: Discovering and improving spaghetti and lasagna processes. In *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*. IEEE, 2011.

[118] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 1st edition, 2011.

[119] Wil M. P. van der Aalst and S. Jablonski. Dealing with workflow change: identification of issues and solutions. *International Journal of Computer Systems Science and Engineering*, 15(5):267–276, September 2000.

[120] Wil M. P. van der Aalst and Maja Pesic. DecSerFlow: Towards a truly declarative service flow language. In *WS-FM*, pages 1–23, 2006.

[121] Wil M. P. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems*. Number 0262720469. The MIT Press, 1 edition, 2002.

[122] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske. Business process management: A survey. In *Business Process Management, International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003, Proceedings*, pages 1–12, 2003.

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

[124] Wil M. P. van der Aalst, H. T. de Beer, and Boudewijn F. van Dongen. Process mining and verification of properties: An approach based on temporal logic. In *OTM Conferences (1)*, volume 3760 of *LNCS*, pages 130–147. Springer, 2005.

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

[126] Wil M. P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23:99–113, 2009. 10.1007/s00450-009-0057-9.

[127] Wil M. P. van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, R. P. Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos C. A. M. Buijs, Andrea Burattin, Josep Carmona, Malú Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano de Leoni, Pavlos Delias, Boudewijn F. van Dongen, Marlon Dumas, Schahram Dustdar, Dirk Fahland, Diogo R. Ferreira, Walid Gaaloul, Frank van Geffen, Sukriti Goel, Christian W. Günther, Antonella Guzzo, Paul Harmon, Arthur H. M. ter Hofstede, John Hoogland, Jon Espen Ingvaldsen, Koki Kato, Rudolf Kuhn, Akhil Kumar, Marcello La Rosa, Fabrizio Maria Maggi, Donato Malerba, R. S. Mans, Alberto Manuel, Martin McCreesh, Paola Mello, Jan Mendling, Marco Montali, Hamid R. Motahari Nezhad, Michael zur Muehlen, Jorge Munoz-Gama, Luigi Pontieri, Joel Ribeiro, Anne Rozinat, Hugo Seguel Pérez, Ricardo Seguel Pérez, Marcos Sepúlveda, Jim Sinur, Pnina Soffer, Minseok Song, Alessandro Sperduti, Giovanni Stilo, Casper Stoel, Keith D. Swenson, Maurizio Talamo, Wei Tan, Chris Turner, Jan Vanthienen, George Varvaressos, Eric Verbeek, Marc Verdonk, Roberto Vigo, Jianmin Wang, Barbara Weber, Matthias Weidlich, Ton Weijters, Lijie Wen, Michael Westergaard, and Moe Thandar Wynn. Process mining manifesto. In *Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I*, pages 169–194, 2011.

[128] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. Replaying history on process models for conformance

checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, March 2012.

[129] Wil M. PP van der Aalst, Vladimir Rubin, HMW Verbeek, Boudewijn F van Dongen, Ekkart Kindler, and Christian W Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software & Systems Modeling*, 9(1):87–111, 2010.

[130] Boudewijn F. van Dongen, Ana Karla Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and Wil M. P. van der Aalst. The ProM Framework: A New Era in Process Mining Tool Support. In Gianfranco Ciardo and Philippe Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2005.

[131] Boudewijn F. van Dongen, Remco M. Dijkman, and Jan Mendling. Measuring similarity between business process models. In Janis A. Bubenko Jr., John Krogstie, Oscar Pastor, Barbara Pernici, Colette Rolland, and Arne Sølvberg, editors, *Seminal Contributions to Information Systems Engineering*, pages 405–419. Springer, 2013.

[132] Maikel L. van Eck, Xixi Lu, Sander J. J. Leemans, and Wil M. P. van der Aalst. PM ^2 : A process mining project methodology. In *Advanced Information Systems Engineering - 27th International Conference, CAiSE 2015, Stockholm, Sweden, June 8-12, 2015, Proceedings*, pages 297–313, 2015.

[133] H. M. W. Verbeek, Joos C. A. M. Buijs, B.F. van Dongen, and W. M. P. van der Aalst. XES, XESame, and ProM 6. In Wil M. P. van der Aalst, John Mylopoulos, Norman M. Sadeh, Michael J. Shaw, Clemens Szyperski, Pnina Soffer, and Erik Proper, editors, *Information Systems Evolution*, volume 72 of *LNBIP*, pages 60–75. Springer Verlag, 2011.

[134] Hagen Völzer. A new semantics for the inclusive converging gateway in safe processes. In *Business Process Management - 8th International Conference, BPM 2010, Hoboken, NJ, USA, September 13-16, 2010. Proceedings*, pages 294–309, 2010.

[135] M. Weidlich, R. M. Dijkman, and J. Mendling. The ICoP Framework: Identification of Correspondences between Process Models. In *CAiSE 2010*, volume 6051 of *LNCS*, pages 483–498. Springer, 2010.

[136] Matthias Weidlich, Mathias Weske, and Jan Mendling. Change propagation in process models using behavioural profiles. In

2009 IEEE International Conference on Services Computing (SCC 2009), 21-25 September 2009, Bangalore, India, pages 33–40, 2009.

[137] Matthias Weidlich, Felix Elliger, and Mathias Weske. Generalised computation of behavioural profiles based on petri-net unfoldings. In *Web Services and Formal Methods - 7th International Workshop, WS-FM 2010, Hoboken, NJ, USA, September 16-17, 2010. Revised Selected Papers*, pages 101–115, 2010.

[138] Matthias Weidlich, Artem Polyvyanyy, Nirmit Desai, Jan Mendling, and Mathias Weske. Process compliance analysis based on behavioural profiles. *Information Systems*, 36(7):1009 – 1025, 2011.

[139] Matthias Weidlich, Artem Polyvyanyy, Jan Mendling, and Mathias Weske. Causal behavioural profiles - efficient computation, applications, and evaluation. *Fundam. Inform.*, 113(3-4): 399–435, 2011.

[140] Matthias Weidlich, Remco Dijkman, and Mathias Weske. Behaviour Equivalence and Compatibility of Business Process Models with Complex Correspondences. *ComJnl*, 2012.

[141] Matthias Weidlich, Tomer Sagi, Henrik Leopold, Avigdor Gal, and Jan Mendling. Making process model matching work. In *Business Process Management - 11th International Conference, BPM 2013, Proceedings*, LNCS. Springer, 2013.

[142] A. J. M. M. Weijters and J. T. S. Ribeiro. Flexible heuristics miner (fhm). In *CIDM*, pages 310–317. IEEE, 2011.

[143] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition*. Springer, 2nd edition, 2012.

[144] Roel J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer-Verlag, 2014.

[145] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2nd edition, 2005.

[146] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer-Verlag Berlin Heidelberg, 2012.

[147] Karsten Wolf. Generating petri net state spaces. In *Petri Nets and Other Models of Concurrency - ICATPN 2007, 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007, Siedlce, Poland, June 25-29, 2007, Proceedings*, pages 29–42, 2007.

[148] Peter Y. H. Wong and Jeremy Gibbons. A process semantics for BPMN. In *Formal Methods and Software Engineering, 10th International Conference on Formal Engineering Methods, ICFEM 2008, Kitakyushu-City, Japan, October 27-31, 2008. Proceedings*, pages 355–374, 2008.

DECLARATION

I hereby confirm that I have authored this thesis independently and without use of others than the indicated sources. All passages which are literally or in general matter taken out of publications or other sources are marked as such. I am aware of the examination regulations and this thesis has not been previously submitted elsewhere.

Potsdam, Germany, August 2015

Thomas Baier