# PROBABILISTIC ESTIMATION OF UNOBSERVED PROCESS EVENTS

ANDREAS ROGGE-SOLTI

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

DISSERTATION

ZUR ERLANGUNG DES GRADES EINES
DOKTORS DER NATURWISSENSCHAFTEN
(DR. RER. NAT.)

JANUARY, 2014

Published online at the
Institutional Repository of the University of Potsdam:
URL http://opus.kobv.de/ubp/volltexte/2014/7042/
URN urn:nbn:de:kobv:517-opus-70426
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-70426

## ABSTRACT

Organizations try to gain competitive advantages, and to increase customer satisfaction. To ensure the quality and efficiency of their business processes, they perform business process management. An important part of process management that happens on the daily operational level is process controlling. A prerequisite of controlling is process monitoring, i.e., keeping track of the performed activities in running process instances. Only by process monitoring can business analysts detect delays and react to deviations from the expected or guaranteed performance of a process instance. To enable monitoring, process events need to be collected from the process environment.

When a business process is orchestrated by a process execution engine, monitoring is available for all orchestrated process activities. Many business processes, however, do not lend themselves to automatic orchestration, e.g., because of required freedom of action. This situation is often encountered in hospitals, where most business processes are manually enacted. Hence, in practice it is often inefficient or infeasible to document and monitor every process activity. Additionally, manual process execution and documentation is prone to errors, e.g., documentation of activities can be forgotten. Thus, organizations face the challenge of process events that occur, but are not observed by the monitoring environment. These *unobserved process events* can serve as basis for operational process decisions, even without exact knowledge of when they happened or when they will happen. An exemplary decision is whether to invest more resources to manage timely completion of a case, anticipating that the process end event will occur too late.

This thesis offers means to reason about unobserved process events in a probabilistic way. We address decisive questions of process managers (e.g., "when will the case be finished?", or "when did we perform the activity that we forgot to document?") in this thesis. As main contribution, we introduce an advanced probabilistic model to business process management that is based on a stochastic variant of Petri nets. We present a holistic approach to use the model effectively along the business process lifecycle. Therefore, we provide techniques to *discover* such models from historical observations, to *predict* the termination time of processes, and to *ensure quality* by missing data management. We propose mechanisms to *optimize* configuration for monitoring and prediction, i.e., to offer guidance in selecting important activities to monitor. An implementation is provided as a proof of concept. For evaluation, we compare the accuracy of the approach with that of state-of-the-art approaches using real process data of a hospital. Additionally, we show its more general applicability in other domains by applying the approach on process data from logistics and finance.

# ZUSAMMENFASSUNG

Unternehmen versuchen Wettbewerbsvorteile zu gewinnen und die Kundenzufriedenheit zu erhöhen. Um die Qualität und die Effizienz ihrer Prozesse zu gewährleisten, wenden Unternehmen Geschäftsprozessmanagement an. Hierbei spielt die Prozesskontrolle im täglichen Betrieb eine wichtige Rolle. Prozesskontrolle wird durch Prozessmonitoring ermöglicht, d.h. durch die Überwachung des Prozessfortschritts laufender Prozessinstanzen. So können Verzögerungen entdeckt und es kann entsprechend reagiert werden, um Prozesse wie erwartet und termingerecht beenden zu können. Um Prozessmonitoring zu ermöglichen, müssen prozessrelevante Ereignisse aus der Prozessumgebung gesammelt und ausgewertet werden.

Sofern eine Prozessausführungsengine die Orchestrierung von Geschäftsprozessen übernimmt, kann jede Prozessaktivität überwacht werden. Aber viele Geschäftsprozesse eignen sich nicht für automatisierte Orchestrierung, da sie z.B. besonders viel Handlungsfreiheit erfordern. Dies ist in Krankenhäusern der Fall, in denen Geschäftsprozesse oft manuell durchgeführt werden. Daher ist es meist umständlich oder unmöglich, jeden Prozessfortschritt zu erfassen. Zudem ist händische Prozessausführung und -dokumentation fehleranfällig, so wird z.B. manchmal vergessen zu dokumentieren. Eine Herausforderung für Unternehmen ist, dass manche Prozessereignisse nicht im Prozessmonitoring erfasst werden. Solch *unbeobachtete Prozessereignisse* können jedoch als Entscheidungsgrundlage dienen, selbst wenn kein exaktes Wissen über den Zeitpunkt ihres Auftretens vorliegt. Zum Beispiel ist bei der Prozesskontrolle zu entscheiden, ob zusätzliche Ressourcen eingesetzt werden sollen, wenn eine Verspätung angenommen wird.

Diese Arbeit stellt einen probabilistischen Ansatz für den Umgang mit unbeobachteten Prozessereignissen vor. Dabei werden entscheidende Fragen von Prozessmanagern beantwortet (z.B. “Wann werden wir den Fall beenden?”, oder “Wann wurde die Aktivität ausgeführt, die nicht dokumentiert wurde?”). Der Hauptbeitrag der Arbeit ist die Einführung eines erweiterten probabilistischen Modells ins Geschäftsprozessmanagement, das auf stochastischen Petri Netzen basiert. Dabei wird ein ganzheitlicher Ansatz zur Unterstützung der einzelnen Phasen des Geschäftsprozesslebenszyklus verfolgt. Es werden Techniken zum *Lernen* des probabilistischen Modells, zum *Vorhersagen* des Zeitpunkts des Prozessendes, zum *Qualitätsmanagement* von Dokumentationen durch Erkennung fehlender Einträge, und zur *Optimierung* von Monitoringkonfigurationen bereitgestellt. Letztere dient zur Auswahl von relevanten Stellen im Prozess, die beobachtet werden sollten. Diese Techniken wurden in einer quelloffenen prototypischen Anwendung implementiert. Zur Evaluierung wird der Ansatz mit existierenden Alternativen an echten Prozessdaten eines Krankenhauses gemessen. Die generelle Anwendbarkeit in weiteren Domänen wird examplarisch an Prozessdaten aus der Logistik und dem Finanzwesen gezeigt.

# PUBLICATIONS

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

- 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 Lecture Notes in Computer Science, pages 389–403. Springer Berlin Heidelberg, 2013.
- Andreas Rogge-Solti, Wil M.P. van der Aalst, Mathias Weske. Discovering Stochastic Petri Nets with Arbitrary Delay Distributions From Event Logs. In *Proceedings of the 9th International Workshop on Business Process Intelligence 2013*, BPM workshops (to appear).
- Andreas Rogge-Solti, Ronny S. Mans, Wil M. P. van der Aalst, and Mathias Weske. Improving Documentation by Repairing Event Logs. In *The Practice of Enterprise Modeling*, volume 165 of Lecture Notes in Business Information Processing, pages 129–144. Springer Berlin Heidelberg, 2013.
- Andreas Rogge-Solti and Mathias Weske. Enabling Probabilistic Process Monitoring in Non-automated Environments. In *Enterprise, Business Process and Information Systems Modeling*, volume 113 of Lecture Notes in Business Information Processing, pages 226–240. Springer Berlin Heidelberg, 2012.
- Andreas Rogge-Solti, Nico Herzberg, and Luise Pufahl. Selecting Event Monitoring Points for Optimal Prediction Quality. In *EMISA*, volume 206 of GI-Edition - Lecture Notes in Informatics (LNI), pages 39–52, 2012.
- Nico Herzberg, Matthias Kunze, Andreas Rogge-Solti. Towards Process Evaluation in Non-Automated Process Execution Environments. In *Proceedings of the 4th Central-European Workshop on Services and their Composition (ZEUS 2012)*. pages 96–102, CEUR-WS.org, 2012,

# ACKNOWLEDGEMENTS

I could not have written this thesis without the support of several people. Foremost, I would like to thank my supervisor Mathias Weske, who guided me in my years at the HPI in many discussions with a professional research spirit. It was a strenuous process for me to find my particular PhD research topic, but in hindsight I am grateful for the freedom that I enjoyed in this matter. I feel that I learned a lot by going through this phase.

I would also like to thank Hajo Reijers and Ulf Leser for agreeing to review this thesis. Additionally, Ulf helped me with his advice in periodic discussion meetings in the role of a mentor in SOAMED.

My colleagues (in order of our sitting distance) Matthias, Anne, Katya, Nico, Rami, Oleh, Andreas, Marcin, Luise, Thomas, and Katrin were always willing to discuss ideas, and created a friendly working environment that I enjoyed very much. Here, I also need to mention my former colleagues Artem, Sergey, Ahmed, Matthias, Alex, Emilian, and Evellin, who shared their experience and ideas with me, when I started my PhD thesis.

In autumn 2012, I was very lucky to be able to go on a research visit to the TU Eindhoven and collaborate with Wil van der Aalst, and with Ronny Mans. The trip was sponsored by the SOAMED research training group. I learned a lot in the short time period, and want to thank especially Wil, Ronny, Christian, Dirk, Michael, and Arya for the inspiring and sometimes very detailed discussions.

Also, I thank Veronika, Matthias, Silvia, and Sie-Youn very much for proofreading parts of this thesis. They invested a lot of their time and helped me improve the readability and understandability of the thesis.

Finally, I would like to thank my family for their support, and Sie-Youn for her encouragement, understanding, and good humor in the more stressful time periods of the dissertation.

# CONTENTS

# LIST OF FIGURES

## LIST OF TABLES



## ACRONYMS

| | |
|---|---|
| BPM | business process management |
| BPMN | Business Process Modeling Language and Notation |
| EMP | event monitoring point |
| GDT_SPN | generally distributed transition stochastic Petri net |
| GSPN | generalized stochastic Petri net |
| IT | information technology |
| MAE | mean absolute error |

| | |
|---|---|
| MAPE | mean absolute percentage error |
| MAR | missing at random |
| MCAR | missing completely at random |
| NMAR | not missing at random |
| PNML | Petri Net Markup Language |
| RMSE | root mean square error |
| SPN | stochastic Petri net |
| UML | Unified Modeling Language |
| XML | eXtensible Markup Language |

# Part I

# BACKGROUND

# 1 INTRODUCTION

Information technology (IT) evolves at an exponential pace as predicted by Moore in 1965 [138]. The rapid development of technology allows the use of increasingly complex and powerful models to capture, analyze, and predict behavior of business processes. Business process management leverages information technology (IT) to automate, monitor, control, and improve business processes. Sophisticated solutions for business processes are on the market already, but these systems are targeted at rather standardized business processes for office workers.

We want to bring process management technologies to hospitals. Hospitals cannot directly adopt well-established methods of business process management (BPM) because of their domain specific requirement of being able to act outside the plan [117]. In this thesis, we consider environments of manual process execution, that is, environments where people perform a process without the orchestration of a process execution engine. One problem that we encounter in such settings is incomplete information. To accurately capture such, we need an approach that enables probabilistic reasoning. As main contribution, we introduce an advanced probabilistic model to BPM that is based on a stochastic variant of Petri nets.

## CHAPTER OUTLINE

This chapter outlines the motivation for the research work presented in this thesis. In Section 1.1, we introduce business process management with a focus on hospital settings. Based on this setting, we derive the problem statement in Section 1.2. We list our contributions in Section 1.3 and the structure of the thesis in Section 1.4.

## 1.1 MOTIVATION

Business process management (BPM) is a key instrument in a company's endeavour to improve the efficiency and the quality of business processes. Latter influence the quality of services and products [206]. Sophisticated methods and tools have been developed to support continuous improvement of business processes along the whole business process lifecycle, see the survey on BPM by van der Aalst et al. [9]. The health care sector is facing an aging population and increasing costs for new drugs and advanced medical devices [204]. Hospitals in Germany, in particular, need to be efficient and reduce costs to remain profitable, as the payment model changed from a length of stay based payment to a payment per case based on diagnosis related groups in 2003 [170].

Regarding business process enactment and analysis, the methods and tools developed for business processes in companies doing mostly office work are not directly applicable in the health care domain, as the focus there is not on products or services, but on human patients [117]. Although there exist best practice models for treatment processes based on evidence [81], each patient is individual and may require individual treatment [195]. Doctors need to be equipped with the freedom to act according to their best knowledge and judgement. Therefore, a one-to-one transfer of workflow engines devised for business process automation to hospitals is infeasible.

Nevertheless, Bates et al. argue that the judicious use of IT in healthcare can help to prevent treatment errors [32]. The authors highlight—among others—the potential use of IT for continuous quality measurement.

The question that arises is: how can hospitals benefit from business process technologies, while still being able to act freely outside predefined processes? A central requirement for business process management is to measure the performance of a process. This triggers the question how to make monitoring systems aware of the current steps that the medical staff perform. There is ongoing research in the direction of ubiquitous computing using sensors, and other devices [193, 47, 33, 69], which aims at automatic tracking of activities in a process. We can expect that this trend continues and more and more information gets accessible to monitoring systems.

Beside these merely technical means to capture process related events, treatment decisions and actions including detailed timing information are recorded by the hospital staff. Medical personnel have the obligation to trace important milestones [27], e.g., in a surgery: when the anesthetic was administered, when the cut was made, etc. These circumstances support the assumption, that at least some events in a treatment process are already (and more will be) available electronically in the hospital information system. Given the case, that models of execution exist, too, we can bind these events to the models at specific points and perform monitoring of the treatment process without a workflow engine.

We can provide estimations for process instances, i.e., we can determine where in the process the patient has been, currently is, and will be (most probably) along the time axis. These estimations can be used to reason about probable resource shortages in a hospital or to predict the length of stay of a patient. Further-

more, these estimations can be used in the case that part of the documentation is missing for a process instance to support early correction of the documentation. Such situations occur in hospitals, because people are responsible to track the processes manually. In this case, we can deduce from the existing documentation and our model *what* the most probable steps in the process were, and *when* they were most probably conducted. Moreover, we can predict the occurrence time of future events based on our probabilistic model.

## 1.2 PROBLEM STATEMENT

The problem statement of this thesis is based on our experiences and gained insights into running processes in hospitals. In the context of the SOAMED Research Training Group[1], we worked on a process elicitation project in the Charité – Universitätsmedizin Berlin [146], and on process monitoring in cooperation with the PIGE project supported by the University of Jena [88]. The problem statement is motivated by settings, where only *incomplete information* of business process execution is available. We encountered such settings in hospitals, where (treatment) processes are performed that contain many activities. These models are performed manually, however, and not all tasks are documented, or connected to the monitoring system. Furthermore, the documentation itself is prone to errors, as it is done manually.

In these settings, we want to create the possibility to reason about unobserved events of process instances that took place in the past, or will take place in the future. However, as we do not have exact information, the best we can do is to use all the prior knowledge and historical observations to create a probabilistic model for *estimating* the events in question.

In process execution settings with manual documentation, it is of interest to reduce the number of times of manual documentation of activities. The goal is to reduce the overhead to the actual work, while still maintaining high prediction accuracy. Similarly, if we set up a monitoring system to capture the performance of a process, we need to decide which events we have to monitor to achieve best prediction accuracy.

The following problems arise in this context:

DISCOVER What is the time performance of the process? How long do certain activities take?

PREDICT When will the current case be finished? When will it reach a certain activity?

ENSURE QUALITY How can we make sure that the documentation is done correctly? How to deal with missing documentation entries?

OPTIMIZE At which points should we require the staff to document steps? Which events should be connected to a monitoring and prediction system?

1 SOAMED homepage: `http://www.ki.informatik.hu-berlin.de/soamed`

## 1.3 SCIENTIFIC CONTRIBUTION

The contribution of this thesis is to provide a probabilistic framework in the context of BPM. This framework is tailored towards the use case of manual process execution and logging, as encountered, e.g., in hospitals. The framework supports the following use cases:

1. Discovery of stochastic performance models [171]
2. Prediction of remaining process durations [177]
3. Reasoning about missing values in process event logs [174, 175]
4. Selection of optimal event monitoring points [172]

To support these use cases, we introduce a stochastic Petri net model to business process management that allows to capture individual activity durations in a flexible way. We collect challenges that occur when applying this particular model and provide respective solutions. The usefulness of this model is evaluated with industry case studies of the motivating domain (i.e., on a hospital surgery process), but also on the domains of logistics and finance.

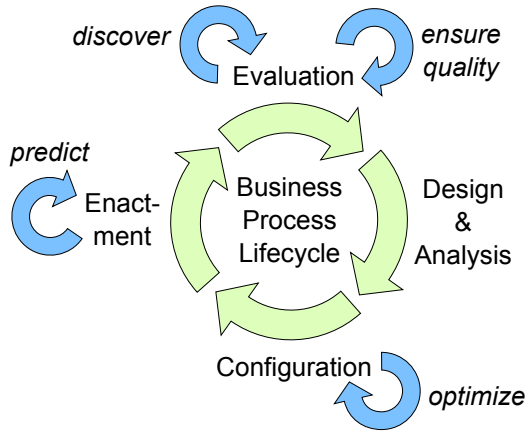



Figure 1: Contributions along the business process lifecycle. We attached the contributions to the business process management lifecycle depicted in the center of the figure. We contribute to the following phases: *Evaluation*, *Configuration*, and *Enactment*.

Figure 1 presents the main contributions of the thesis along the business process lifecycle. This figure shall guide us through the main part of the thesis.

## 1.4 THESIS STRUCTURE

The thesis is divided into three parts.

PART I The first part is introductory. It presents the background of the thesis, and its motivation. In Chapter 2, we introduce fundamental concepts that are required for this thesis.

PART II In the main part of the thesis, we explore each contribution in detail. The contributions constitute the framework for probabilistic estimation of unobserved process events. The contributions are aligned with the four use cases listed in the previous section. In Chapter 3, we show how we can *discover* probabilistic models from historical observations. Then, in Chapter 4, we use the discovered model to *predict* the remaining time of a running process instance. In Chapter 5, we investigate how the probabilistic model can assist to *ensure quality* of documentation in manual processes. The last contribution is to the *configuration* phase of the business process lifecycle. In Chapter 6, we present a method to *optimize* the selection of monitoring points that are most relevant for monitoring and prediction.

PART III In the final part, we transfer the concepts and techniques to case studies. In Chapter 7, we discuss the implementation of the concepts. We evaluate the techniques on real data from a hospital in Chapter 8. Moreover, we show the applicability of the techniques to process data from logistics and finance. Finally, in Chapter 9, we conclude the thesis and present an overview of unresolved issues and next steps.

# 2 REQUIRED FUNDAMENTALS

In this chapter, we present existing formalisms and approaches, which are fundamental for this thesis. We provide definitions of the concepts here. Later, we will use them in the proposed framework for probabilistic estimation of unobserved process events.

### CHAPTER OUTLINE

We start in Section 2.1 with the introduction to business process management (BPM), which is ubiquitous in this work. Then, we investigate Petri nets and stochastic extensions capturing temporal aspects of Petri nets in Section 2.2. Subsequently, we give an introduction to process mining in Section 2.3. We overview existing monitoring architectures in Section 2.4. Thereby, we address the question of correlation and mapping between events and process models. We conclude with basic concepts from probability theory, and probabilistic models in Section 2.5.

## 2.1 BUSINESS PROCESS MANAGEMENT

In a globally competing environment, organizations strive to improve their efficiency and effectiveness [84, 206]. Tools and techniques have emerged to support the task of managing business processes. The discipline of BPM is best introduced along the business process lifecycle.

### *Business Process Lifecycle*

The business process lifecycle [9] provides a high level view on the phases of a business process. The lifecycle consists of the following phases: (1) evaluation, (2) design and analysis, (3) configuration, and (4) enactment of business processes [206], as depicted in Figure 2.

In the *evaluation* phase the as-is business processes of an organization are examined. Thereby, techniques such as process mining [3] can be used to gain insights into the running processes. In the *design and analysis* phase of business process management, the processes are identified, and experts capture the to-be processes in the form of business process models. If business process models are already available, redesign [84, 167] can be performed. The redesigned process models can be checked for correctness [1], and simulation tools [97] can be used to check, whether the changes improve the as-is processes.

When a satisfactory business process model is selected for adoption, the process has to be deployed to the process environment in the *configuration* phase. Depending on the execution environment, this phase can result in briefing employees to follow the new specifications, or in a configuration of a workflow engine [207] that controls the execution of a business process automatically. Latter is only possible when the environment allows for automation of business processes. For example in healthcare, automation proves to be especially challenging [117]. There are cases, in which processes are well understood and exhibit small variance and high frequency of occurrence; these cases are good candidates

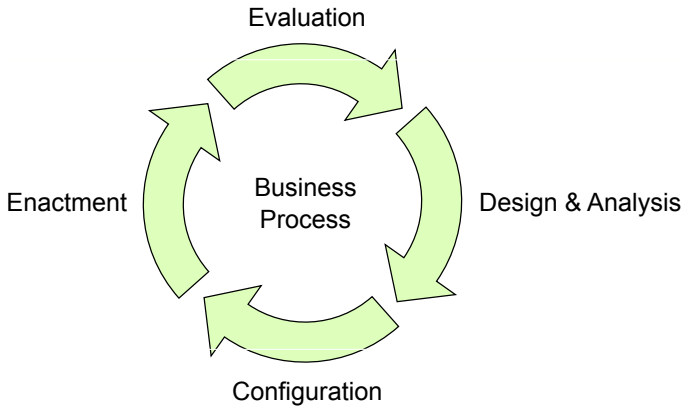



Figure 2: Business process lifecycle. The business process lifecycle forms a continuous process that focuses on maintaining and improving the performance and quality of a business process. It consists of four phases: (1) evaluation, (2) design and analysis, (3) configuration, and (4) enactment, cf. [206].

for automation by workflow engines. However, in many process enactment environments, the use of workflow engines is infeasible. Infeasibility can be due to high flexibility requirements, complexity or rare occurrence. Then cost of implementing a workflow engine is not justified by the efficiency gained through automated orchestration.

In the *enactment* phase, which closes the business process management lifecycle, the operation of the business process takes place. We shall take a more detailed look at this phase in the next section. In particular, we shall investigate execution and monitoring.

### *Execution and Monitoring*

The process enactment phase is where the daily business of organizations is conducted. Customers request specific products or services of a company, which in turn delivers these through business processes. The most interesting aspect of business process execution, at least from a management point of view, is whether the process is executed efficiently (i.e., in short time and with economical resource usage), and whether their execution conforms to the specified models. Business process analytics [145] offers tools to ensure smooth operation, e.g., by identifying bottlenecks.

To conduct business process analytics, we need to be able to measure the performance of a business process. We use the term *business process monitoring* for the task of measuring progress of process instances during their execution. In the remainder of this thesis, we refer to this task as *process monitoring* or simply *monitoring*. The aim of process monitoring is to provide decision support during enactment [57]. We only can ensure the quality of process performance, if we have insights into the running processes. Therefore, we need to measure them. Monitoring enables us to react appropriately to changes in the process, e.g., an increase in demand. An example shall illustrate this: If we detect that, currently, there are a lot of cases arriving, and only few cases completing, we know that there is a rising number of cases. In this situation, process managers can invest more resources (e.g., request additional assisting specialists) to increase the chance of timely completion of cases and avoid long waiting periods.

### *Business Process Models*

Business processes are the central, value-generating processes in companies that capture activities and their relations to provide products or services to clients [206]. Modeling of business processes is essential for business process management, as models allow us to capture the essential components of a business process and put them into relation to each other.

According to a study by Indulska et al. [94], the five top-ranked perceived benefits of business process modeling are (1) process improvement, (2) understanding, (3) communication, (4) model-driven process execution, and (5) process performance measurement. Obviously, every organization tries to obtain

these benefits. To do so, however, business process models need to be of good quality [133] and captured in an appropriate modeling language.

There exist several modeling techniques to capture business process models, cf. the survey in [121]. For example, flow charts have been used to model algorithms since the introduction of computers, cf. [78]. But also UML activity diagrams [151], event driven process chains [108], or the Business Process Modeling Language and Notation (BPMN) [150] are candidates for modeling business processes. Noteworthy candidates from academia are Petri nets [159, 2], YAWL [7], and ADEPT [165]. Petri nets are abundantly used with various extensions, e.g., high-level Petri nets [22], or colored Petri nets [190, 98] have been proposed [99, 134]. See also [188] for an overview on the use of Petri nets for workflow modeling. We shall return to Petri nets in the next section. Currently, the BPMN is the established de facto standard for modeling business processes [192].

An example business process model in BPMN is depicted in Figure 3. The process model captures a treatment process as encountered in hospitals performing surgeries. Important in this example are the activities and their relations. First, the patient is registered before being examined. Note the exclusive choice after the examination of the patient. Here, the patient either is transferred to another hospital, or the surgery will be performed after a scheduling activity. Afterwards, the patient will be taken care of in stationary care, until the patient can be released.

In the given example in Figure 3, we only encounter a small number of modeling constructs. The BPMN modeling language offers a wide range of available shapes to model business processes, see Figure 4 for an overview of the basic shapes.

Throughout this thesis, we will use only a subset of BPMN models, that we can translate into a Petri net [159] representation. This simplification is justified by empirical research work by zur Mühlen and Recker [144] that shows that most models in practice do not use the full set of modeling constructs defined in the BPMN language specification [150]. That is, the basic structural modeling constructs encountered in Petri nets suffice for capturing most business processes. Additionally, the use of Petri nets can also be directly used in workflow management as proposed in [2]. Several mappings for the most common workflow patterns of different process modeling languages into Petri nets have been collected by Lohmann et al. in [120]. For models captured in BPMN, a translation to Petri nets is presented by Dijkman et al. [59]. Therefore, we assume that busi-

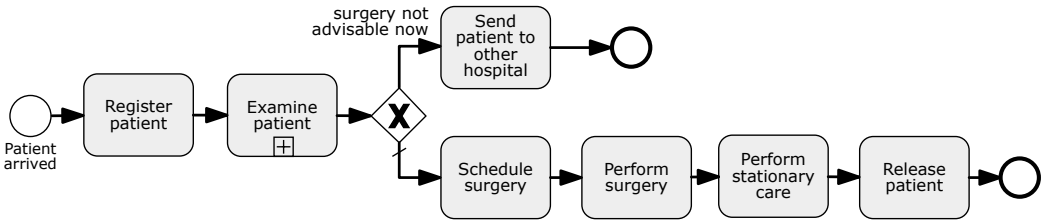



Figure 3: Abstract surgery BPMN process model. This simple example illustrates a mostly sequential treatment process that describes the activities in the course of a treatment containing a surgery.

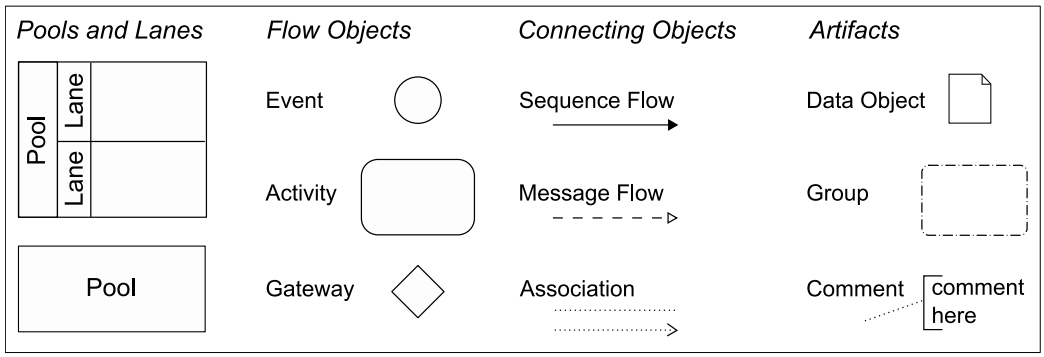



Figure 4: Overview of basic shapes available in BPMN. Figure from [146] (cf. [150]). Pools and lanes are used to group activities and events by organizations and responsibilities. Flow objects capture the main modeling elements and are connected by sequence flows. Message flows represent exchange of information with other organizations. Associations relate the flow objects to artifacts. These basic shapes can be extended by markers, resulting in constructs like, *throwing intermediate message* events, *looping service-task*, or *event-based* gateways). The standard lists 48 valid combination of events [150].

ness process models are available directly—or through translation—in a Petri net representation.

## 2.2 STOCHASTIC PETRI NETS

Before we consider stochastic extensions of Petri nets, we first introduce the basic Petri net model. The Petri net formalism dates back to the seminal thesis of Petri in 1962 [159]. We rely on the original definition of Petri nets [159], which is defined as follows.

**Definition 1 (Petri Net)** *A Petri net is a tuple* $PN = (P, T, F, M_0)$ *where:*

- $P$ *is a set of places,*
- $T$ *is a set of transitions,*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of connecting arcs representing flow relations,*
- $M_0 \in P \to \mathbb{N}_0$ *is an initial marking.*

Petri nets offer a simple set of basic elements that form a directed bipartite graph of places and transitions. A state of a Petri net is determined by the marking $(M : P \to \mathbb{N}_0)$ of a net, which is specifying the number of *tokens* on the places. For markings $M$, with $M(p) \leqslant 1$ for any $p \in P$, we use the shorthand set notation of places with a token in the marking, e.g., $\{p_1, p_4\}$ is the marking $M$ with $M(p_1) = 1$ and likewise $M(p_4) = 1$ and all other places are empty (i.e., $\forall p \in P \setminus \{p_1, p_4\} : M(p) = 0$). Transitions can have input and output places, i.e., a transition $\mathbf{t} \in T$ has input places $\bullet\mathbf{t} = \{p \in P \mid (p, \mathbf{t}) \in F\}$, and output places $\mathbf{t}\bullet = \{p \in P \mid (\mathbf{t}, p) \in F\}$. Transitions capture changes of states. They also have a clear firing semantic, i.e., transitions are enabled, if all their input places contain at least one token. When an enabled transition fires, it removes a token from each of its input places and adds a token to each of its output places. This simple firing rule allows us to specify sequential, exclusive, parallel, and recurring

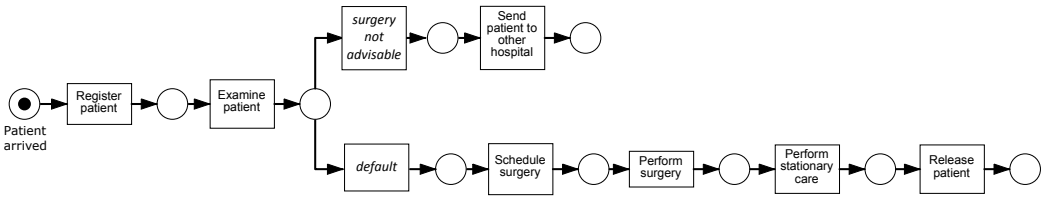



Figure 5: Abstract surgery Petri net model. Here a possible translation of the model in Figure 3 to a Petri net is depicted. Transitions are represented as rectangles, and places as circles.

relations between transitions, by connecting places and transitions in different patterns in the Petri net. For example, two transitions with the same input place represent an exclusive choice, because the transition first firing takes away the token from the competing transition, disabling the latter.

Another benefit of Petri nets is their mathematical flavor, which allows for various analysis methods [140, 168]. Petri nets are used, among others, for verification of workflow systems [1]. In fact, a considerable part of research in the BPM area deals with verification of *soundness* properties of business process models [1], or the compliance to a given set of business rules [35, 123, 26].

Soundness is defined for workflow nets—a special class of Petri nets that has a dedicated input place $p_i$ with no incoming arcs (i.e., $\nexists \mathbf{t} \in T \mid (\mathbf{t}, p_i) \in F$) and an output place $p_o$ with no outgoing arcs (i.e., $\nexists \mathbf{t} \in T \mid (p_o, \mathbf{t}) \in F$), and all places and transitions are on a path between these two places. Soundness requires that

- the final marking is reachable from every intermediate marking that can be reached from the initial marking.
- for each transition there exists a firing sequence that includes the transition and starts with the initial marking.
- if the final marking is reached (i.e., the output place $p_o$ of the workflow net has a token), no other place may still contain tokens.

Besides guaranteeing deadlock-freedom, soundness implies boundedness (i.e., there are only a finite number of markings that can be reached from the initial marking) [1].

An example translation of the BPMN model presented in Figure 3 is depicted in Figure 5. By convention, tansitions are rectangular and places are represented as circles. Note that due to the bipartite nature of Petri nets the model has more nodes than the original model and the decision to do the surgery is captured as a place followed by two competing transitions (labels italic).

Various kinds of extensions to Petri nets have been proposed in order to capture performance criteria. In his thesis, Ramchandani introduced the time dimension into Petri nets [162]. Timed Petri nets use deterministic timings, and are of limited use to model processes which involve uncertainty. The idea to integrate exponentially distributed timings into Petri nets was independently proposed by Symons [197], Natkin [148], and Molloy [137]. In his PhD thesis [136], Molloy also has shown that these models are isomorphic to Markov chains, which can

be solved analytically. The following definition captures the model of stochastic Petri net (SPN), cf. [137].

**Definition 2 (Stochastic Petri Net)** *A stochastic Petri net is a five-tuple: SPN* $= (P, T, F, M_0, \mathcal{L})$*, where* $(P, T, F, M_0)$ *is the basic underlying Petri net. Additionally,* $\mathcal{L} : T \to \mathbb{R}^+$ *is an assignment of transition rates to transitions.*

The transition rates in SPNs serve as parameters to the exponentially distributed firing durations of each transition. The execution semantics of SPN models is stochastic, i.e., the choice which transition fires in a certain marking is probabilistically made among the enabled transitions based on their transition rates. For example, when a transition $\mathbf{t}_A$ with rate $\mathcal{L}(\mathbf{t}_A) = 2$, is concurrently enabled with another transition $\mathbf{t}_B$, that has a firing rate of $\mathcal{L}(\mathbf{t}_B) = 1$, the $\mathbf{t}_A$ fires twice as often, as transition $\mathbf{t}_B$.

Stochastic Petri nets were studied extensively over the years, and numerous approaches were proposed extending the modeling capabilities, while at the same time maintaining analytical [130, 129], or at least numerical [118], tractability. An overview over different classes of stochastic Petri nets is given by Ciardo et al. in [51].

One well-known formalism is provided by Ajmone Marsan et al. [130]; it is called generalized stochastic Petri net (GSPN), and is defined as follows.

**Definition 3 (Generalized Stochastic Petri Net)** *A generalized stochastic Petri net is a seven-tuple, GSPN* $= (P, T, \mathcal{P}, \mathcal{W}, F, M_0, \mathcal{L})$*, where* $(P, T, F, M_0)$ *is the basic underlying Petri net. Additionally:*

- *The set of transitions* $T = T_i \cup T_t$ *is partitioned into immediate transitions* $T_i$ *and timed transitions* $T_t$
- $\mathcal{P} : T \to \mathbb{N}_0$ *is an assignment of priorities to transitions, where* $\forall \mathbf{t} \in T_i : \mathcal{P}(\mathbf{t}) \geqslant 1$ *and* $\forall \mathbf{t} \in T_t : \mathcal{P}(\mathbf{t}) = 0$
- $\mathcal{W} : T_i \to \mathbb{R}^+$ *assigns probabilistic weights to immediate transitions*
- $\mathcal{L} : T_t \to \mathbb{R}^+$ *is an assignment of transition rates to timed transitions, specifying the rate parameter of the exponentially distributed firing durations.*

GSPN models have been shown to be isomorphic to Markov chains [130], which can be analyzed efficiently. The topic of analysis methods for Petri nets is further developing, e.g., Katoen recently proposed a novel algorithm to analyze GSPNs as stochastic real-time games [106].

For some processes, we do not want to restrict the transition durations to follow the negative exponential distribution, or be immediate. For example, the duration of a surgery in a hospital treatment process is assumed to follow the normal (or log-normal) distribution [196]. Also there might be activities or events that are not random, but deterministic, as for example time-outs, which cannot be captured properly by an exponential distributions or their variants.

Other researchers, e.g., Dugan and colleagues, found the exponentially distributed timings too restrictive [64, 39], and proposed extensions of stochastic Petri net models to allow arbitrary firing delays [64]. These more permissive models are called extended stochastic Petri nets [64], non-Markovian stochastic Petri nets [75, 90], or generally distributed transition stochastic Petri nets [127,

39]. Although offering more flexibility in model specification, these models are in general not efficiently analyzable. Discrete event simulation has to be used for their analysis [64] or, given certain assumptions (e.g., block-structuredness), they can be approximated numerically [166]. The use of GSPN models remains popular, because of their simple characteristics [105].

Several researchers have devoted work on pushing the limit of analytical tractability of extended models [64, 75, 50, 218]. A special generalization of stochastic Petri nets—Markov regenerative Stochastic Petri nets—is shown to be tractable with numerical analysis [50]. Moreover, tool support with implementation of these numerical analysis methods is available [219, 218]. However, latter numerical tractability requires that in each marking only one transition with generally distributed timing is available. A review of different approaches for the analysis of non-Markovian stochastic Petri nets is given by German in [74].

For our purposes, we extend the widely known definition of GSPNs provided in [130], by allowing durations of the timed transitions to be generally distributed. In terms of the categorization proposed in [51], we use the most general—or most permissive—model class of stochastic Petri nets.

**Definition 4 (Generally Distributed Transition Stochastic Petri Net)** *A generally distributed transition stochastic Petri net (*GDT_SPN*) is a seven-tuple:*
$\mathsf{GDT\_SPN} = (P, T, \mathcal{P}, \mathcal{W}, F, M_0, \mathcal{D})$*, where* $(P, T, F, M_0)$ *is the basic underlying Petri net. Additionally:*

- *The set of transitions* $T = T_i \cup T_t$ *is partitioned into immediate transitions* $T_i$ *and timed transitions* $T_t$
- $\mathcal{P} : T \to \mathbb{N}_0$ *is an assignment of priorities to transitions, where* $\forall \mathbf{t} \in T_i : \mathcal{P}(\mathbf{t}) \geqslant 1$ *and* $\forall \mathbf{t} \in T_t : \mathcal{P}(\mathbf{t}) = 0$
- $\mathcal{W} : T_i \to \mathbb{R}^+$ *assigns probabilistic weights to the immediate transitions*
- $\mathcal{D} : T_t \to D$ *is an assignment of arbitrary probability distributions* $D$ *to timed transitions, reflecting the durations of the corresponding activities.*

Besides this choice for modeling, there are also similar models used for capturing performance in workflows. In his thesis, Reijers defined a similar class called Stochastic Workflow net [166], which is a workflow net enriched with stochastic timing parameters. The main difference to GDT_SPN models is that he uses discretized timing specifications, and restricts models to be block structured. A comprehensive overview on the historical evolution of stochastic Petri nets is given by Puliafito and Telek in [161]. Further, Balbo provides a more recent introduction to the topic in [30].

One key characteristic of the GDT_SPN model, that our work shares with other stochastic models, is the *independence assumption*. That is that transition durations are independent from one another. This assumption and using a distribution family in the model, that is closed under the operations of summation and maximization, allows exact analysis techniques as available for GSPN. In practice, this assumption might not always be valid, and models assuming independence yield poor a explanation of the real process behavior.

An extreme example helps to illustrate this point. Imagine a process in which two activities A and B are normally distributed—both with a mean duration of

5 and a variance of 1. If the two activities are independent, the sum of A and B is a new random variable with the mean of $5 + 5 = 10$ and a variance of $1 + 1 = 2$. This is what a GDT_SPN model would return for the resulting durations as well. If instead the two activities A and B would be correlated positively with a correlation coefficient of 1 (i.e., the duration of activity A being in a perfect linear relationship with the duration of activity B) the resulting sum of the durations would be normally distributed with a mean of 10 and a variance of 4. On the other hand, if the activities were in a perfect negative correlation (with coefficient $-1$), the resulting mean of the sum of A and B would be 10 and the variance would be canceled to 0. If such strong correlations exist in a process, models with the independence assumption are not the models of choice.

In Chapter 8, we shall come back to this assumption and evaluate the GDT_SPN model in three case studies to see if the independence assumption is too strong, or if the model can still capture the real behavior well.

## 2.3 PROCESS MINING

In this thesis, we assume that business process models are available and capture the behavior of a business process accurately. Even if no process models exist, the assumption that models exist should not be regarded as a limitation of this thesis, because numerous *process mining* techniques can be used to discover process models from event logs, cf. the survey in [11], or the textbook by van der Aalst [3].

### *Events and Event Logs*

We first introduce the terms *events* and *event logs* that are at the core of process mining. Event logs are a central artifact in process mining, as they contain the information of interest in raw form. Typically, various sources can generate events, e.g., information systems, workflow management systems, and enterprise resource planning systems.

The term *event* is ambiguous, as it is used in many similar settings with different meaning, e.g., in the logistics community, an event marks exceptional occurrences that require adjustment of a plan [76]. In daily life, events might specify certain happenings, such as concerts, the visit of a famous person, etc. Luckham defines events as something that happens or occurs and might change the current state of the system [122].

In this work, we understand events as indicators for progress in business processes. More specifically, an event indicates that an activity of a corresponding process instance has changed its state, e.g., was completed. We largely abstract from the technological details of how events are collected from different sources in this thesis, and show but a few example architectures for this purpose in the next section. Therefore, we work at the abstraction level of *event logs* that capture relevant events of a business process grouped by the corresponding case, as also defined in process mining [3].

**Definition 5 (Event Log)** *An event log over a set of activities* $A$ *and time domain TD is defined as* $L_{A,TD} = (E, C, \alpha, \beta, \gamma, \succeq)$*, where:*

- $E$ *is a finite set of events*
- $C$ *is a finite set of cases (process instances),*
- $\alpha : E \to A$ *is a function assigning each event to an activity,*
- $\beta : E \to C$ *is a surjective function assigning each event to a case.*
- $\gamma : E \to TD$ *is a function assigning each event to a timestamp,*
- $\succeq \subseteq E \times E$ *is the succession relation, which imposes a total ordering on the events in E.*

Note that event logs are not restricted to the components in this definition in general. Rather, many information systems keep track of additional aspects of events, e.g., roles, used resources, costs, data. However, in this thesis we focus on the time domain. In practice, correlation of events with cases (i.e., defining the $\beta$ function for each event) is non-trivial, especially if the data is unstructured as in our use cases. Dealing with the correlation issue is out of the scope of this work, however. The interested reader is referred to the seminal work by Agrawal et al. in [18], where the authors describe an algorithm to mine association rules from data. Another interesting approach is offered in the paper by Motahari-Nezad et al. [139], which offers correlation conditions that specify how events will be correlated with activities, and also offers discovery capability of these conditions from an event log. Independently, Musaraj et al. [143] provide a method to identify correlation on message level based on information of timestamps only.

We assume that the environment that generates the events also sets the timestamps (i.e., the $\gamma$ function) of the events. Setting timestamps depends on the domain, while in some cases it might make sense to use the timestamps of event creation, in other cases it might be possible to extract the timestamps from the event sources.

The assignment of events to activities (i.e., the $\alpha$ function in the event log definition), is of interest in this thesis. We shall explain the notion of alignments [15] in the next section.

The process mining manifesto [4] positions process mining in the context of the real world, process models, software systems, and event logs, cf. Figure 6. Process mining can be categorized into the following three dimensions:

1. *discovery*, where a process model is inferred from the event logs that capture execution traces of processes
2. *conformance*, where a process model is compared with observed execution in form of event logs
3. *enhancement*, where a model is enriched with information extracted from event logs

In the following, we briefly introduce these concepts.

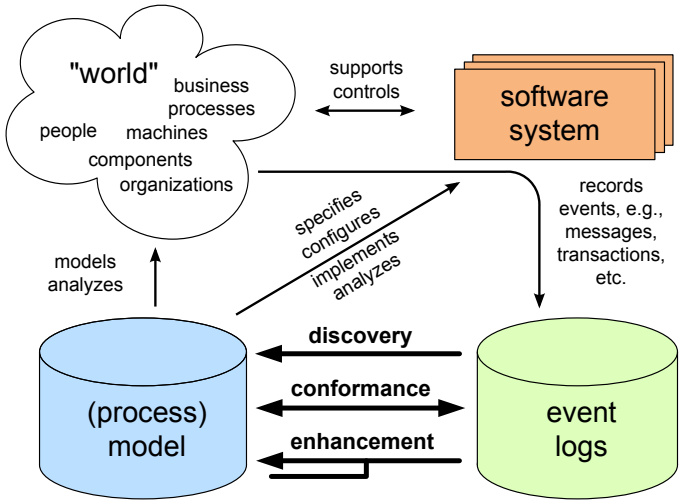



Figure 6: Process mining overview. The three major aspects of process mining are 1) discovery, 2) conformance, and 3) enhancement, cf. [3].

*Discovery of Process Models*

Conceptually, the problem of discovery (i.e., identifying the minimal process model that captures the relationships observed in an event log) is related to the problem of finding a minimum finite-state automaton that is compatible with given data, which has been shown to be NP-hard by Gold in [77]. Besides being computationally challenging, the problem also sparks interest in research and industry due to one of its greatest promises: the automation of the cumbersome task of eliciting process models manually (e.g., with interviews).

The research area of process mining started with the work by Agrawal et al. [17], in which the authors presented an automatic way to derive models from causal dependencies of events gathered from an event log. Since then, a series of mining algorithms were developed: algorithmic approaches, e.g., the *alpha*-algorithm [12], which discovers Petri nets, or the related work by Herbst and Karagiannis [87]. Latter constructs a model reflecting all traces as a Markov chain, and applies iterative merging steps to simplify the model. Additionally, there are techniques using heuristic approaches to tackle the problem, e.g., using genetic algorithms [132].

*Conformance Checking*

To ensure quality of a business process, it is not sufficient to specify how a process should be executed, e.g., in the form of business process models. Rather, it is necessary to test, whether the specified models that capture the desired behavior are executed as specified in practice. Conformance checking is the discipline in process mining that tries to *align* the execution trace of a case to the model in order to detect deviations. It helps to identify parts in the process model that need refinement, or where employees need to adapt their work so that it conforms to the specification.

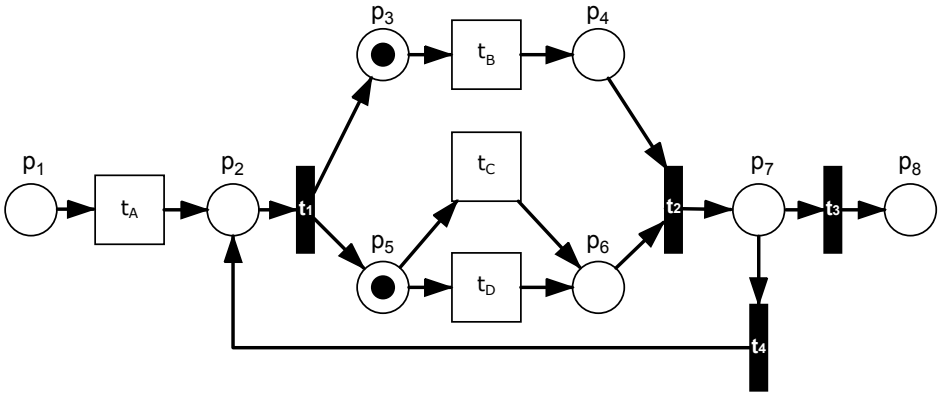



Figure 7: A more abstract Petri net model. This more abstract model serves as example for explaining the alignment notion. It contains sequential, parallel, exclusive, and iterative structural relationships.

First results were achieved by Rozinat and van der Aalst [181, 182], which use a form of replay of a log in a model, and count the number of additionally inserted and removed tokens that are necessary to replay a trace in the Petri net model. In this early work, the traces in the logs are greedily replayed in a model and thus there is no guarantee to find an optimal path through the model. In successive work, Adriansyah et al. introduced the notion of a cost-based alignment in [15]. Latter work guarantees to find a globally optimal alignment [15] that considers the structural properties of the model. Therefore, tt assigns costs to additional steps in either model or trace that cannot be mimicked by the other.

To quantify conformance, the *fitness* measure is used [181]. A fitness value tells us, to what degree the observed cases in the event log follow the behavior prescribed by the model. Note that the measure of fitness used in conformance is usually defined as a value in the range between 0 and 1. Let $L_{A,TD}$ be an event log, and *pn* be a corresponding Petri net model. Then, van der Aalst et al. [5] define fitness as:

$$\text{fitness}(L_{A,TD}, pn) = 1 - \frac{\text{cost of optimal alignment between } L_{A,TD} \text{ and } pn}{\text{maximum cost of log } L_{A,TD} \text{ and model } pn} \quad (1)$$

Usually the costs of synchronous and invisible moves are set to zero and other moves (i.e., log moves and model moves) have costs greater than zero. With such a configuration, this definition of fitness yields a value of 1 for an optimal alignment between model and log (i.e., where the cost of that alignment are zero). If not a single synchronous move can be found between log and model, i.e., the optimal alignment contains the cost of all the model moves and log moves, the fraction in Equation 1 is 1, and this definition yields a fitness value of 0.

We will build on the cost-based alignment technique throughout this thesis. In the following, we offer an intuition, and refer to the report by Adriansyah et al. [16] for the formal details.

To explain the notion of alignments between a process model and an event log, we use the abstract example Petri net model in Figure 7. In this part, we do not focus on the semantics of the model, but on the structure. Therefore, we use

a model that contains sequential relationships ($\mathbf{t}_A$, $\mathbf{t}_B$), parallelism (between $\mathbf{t}_B$ and one of $\mathbf{t}_C$ or $\mathbf{t}_D$), as well as a cycle (returning from $p_7$ to $p_2$).

As convention, we use subscript roman letters for transitions in $T_t$ that reflect activities (e.g., $\mathbf{t}_A$, ..., $\mathbf{t}_D$), while we use subscript numbers for places (e.g., $p_1$, ..., $p_8$), and transitions in $T_i$ for control flow routing (e.g., $\mathbf{t}_1$, ..., $\mathbf{t}_4$).

Figure 8 shows two execution traces ($tr_1$, $tr_2$) of the model depicted in Figure 7. Each event in the trace corresponds to a transition in the net with matching subscript, e.g., event *B* belongs to transition $\mathbf{t}_B$. For this example, we assume that immediate transitions are invisible to the monitoring system (i.e., they do not appear in the log), and all timed transitions are visible.

The cost-based alignment approach aligns each observed trace to a particular path in the model. The challenge is to decide, to which activity instance an event belongs. Note that we are not limited to cases where activity labels in business processes are unique, but even if that was the case, cycles in process models allow us to perform the activities multiple times. Therefore, a trace of a process model can contain events belonging to different instances of the same activity. We assume that a mapping from event types to activity types exists, but we need to identify which occurrence of that activity type to assign an event to. More generally, conformance checking tries to identify missing events, additional events or incorrect ordering of events.

The alignment method synchronizes each trace with the process model in a *product* model [210]. The product of two Petri nets contains both nets and introduces synchronous transitions that stand for simultaneous occurrence of events in both nets. Once pairs of synchronous transitions have been identified in the models, we can create a product by adding a transition for each synchronous pair. The transition is connected to the input places and the output places of the transitions to by synchronized. A formal definition of a product of Petri nets is given in [210]. In this thesis we do not repeat it, but explain this concept with an example.

Figure 9 shows the product model that we can construct for the trace $tr_1$ in Figure 8 and the Petri net model in Figure 7, cf. [16]. Because each event in $tr_1$ is mapped to the corresponding transition by naming convention, we create the synchronous transitions accordingly. In a perfect alignment, we can replay both the model and the log from start to end using only synchronous movements and invisible movements (these are elements in the model that are not reflected in the event logs, e.g., the splitting transition $\mathbf{t}_1$). This is not the case, if an event log does not conform to the execution order dictated by the model.

$$
\begin{aligned}
tr_1 &: \quad \langle\, A, \quad B, \quad D, \quad C, \quad B \,\rangle \\
tr_2 &: \quad \langle\, B, \quad D, \quad D \,\rangle
\end{aligned}
$$

Figure 8: Example event log. This event log refers to the model in Figure 7. The event log contains two traces. The trace $tr_1$ was created in a case where the cycle was traversed twice. The trace $tr_2$ does not conform to the model. (We omitted timestamps, as they are irrelevant to the discussion.)

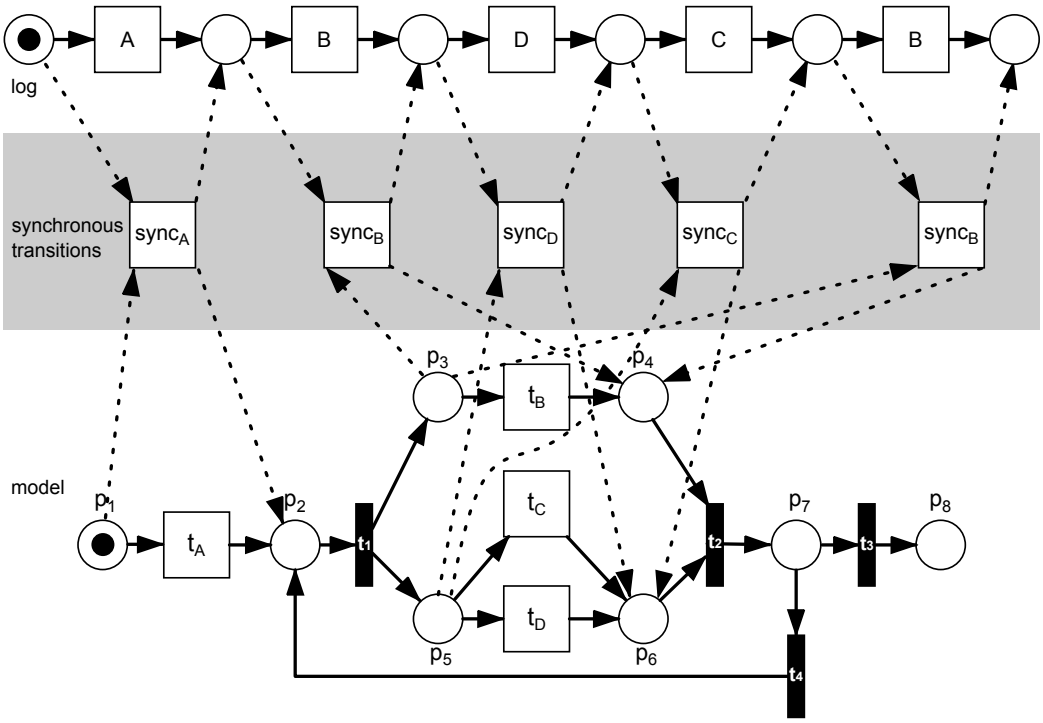



Figure 9: Product of a Petri net model and a trace. The product can be derived automatically by converting a trace (in this case $tr_1$) to a sequential Petri net model. Then appropriate synchronous transitions for each event in that trace are created, that use the union of the input and output places in the model and the trace of the transitions that are synchronized. The arcs of the synchronous transitions are dashed merely for visualization purposes to avoid cluttering the figure.

The idea of cost-based alignments is to set costs for asynchronous moves in both model and log of the product model to penalize alignments containing asynchronous moves. Then, the identification of optimal alignments can be achieved by searching for the shortest paths in the state transition system of the product model [15]. We need to define the final marking of the process model, which in case of workflow nets [1] is one token at the final place. The final marking of the product is denoting the state when both model and trace reached their end. Next, we shall take a look at an example.

*Example Alignments*

Here, we discuss possible alignments for the example model and event log introduced in Figure 7, and Figure 8 respectively. We denote invisible transitions in the alignment with a $\tau$ symbol. Note that trace $tr_2$ does not fit well into the model, and we want to find an optimal alignment between model and log. For this purpose, we reuse the methods developed by Adriansyah et al. in [15], which results in a sequence of movements that *replay* the trace in the model. These movements are either *synchronous moves*, *model moves*, or *log moves*. Figure 10(a) displays a *perfect* alignment for $tr_1$ that consists of synchronous or invisible model moves only. The $\gg$ symbol represents no progress in the replay

| *log* | A | ≫ | B | D | ≫ | ≫ | ≫ | C | B | ≫ | ≫ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *model* | A | τ | B | D | τ | τ | τ | C | B | τ | τ |
| | $t_A$ | $t_1$ | $t_B$ | $t_D$ | $t_2$ | $t_4$ | $t_1$ | $t_C$ | $t_B$ | $t_2$ | $t_3$ |

(a) perfect alignment for trace $tr_1$.

| *log* | ≫ | ≫ | B | D | D | ≫ | ≫ |
|---|---|---|---|---|---|---|---|
| *model* | A | τ | B | D | ≫ | τ | τ |
| | $t_A$ | $t_1$ | $t_B$ | $t_D$ | | $t_2$ | $t_3$ |

(b) a possible alignment for trace $tr_2$.

| *log* | ≫ | ≫ | B | D | ≫ | ≫ | ≫ | D | ≫ | ≫ | ≫ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *model* | A | τ | B | D | τ | τ | τ | D | B | τ | τ |
| | $t_A$ | $t_1$ | $t_B$ | $t_D$ | $t_2$ | $t_4$ | $t_1$ | $t_C$ | $t_B$ | $t_2$ | $t_3$ |

(c) another possible alignment for trace $tr_2$.

Figure 10: Three possible alignments. These alignments could be selected (a) for trace $tr_1$, or (b) and (c) for trace $tr_2$ in Figure 8.

on either side, e.g., the first step in the alignment in Figure 10(b) is a model move. Observe that by replaying the alignment in the product net in Figure 9, we reach both the end of the process model and the trace $tr_1$.

For trace $tr_2$ there exist multiple alignments, of which two are depicted in Figures 10(b) and 10(c). In fact, for the Petri net model in Figure 7 there exist an infinite number of alignments, as the model contains a cycle that can be traversed an arbitrary number of times. However, each additional iteration results in two additional model moves, and three invisible model moves per iteration. The cost based alignment approach in [15] makes sure that alignments containing unnecessary moves get penalized by higher cost and get excluded from the optimal alignments. An alignment provides a deterministic firing sequence in the model replaying the traces in an optimal way. But in general, there can be multiple optimal alignments for a particular trace.

It is possible to set costs to model and log moves individually, to set preferences to certain log moves or model moves. For example, one could set the costs for log moves higher than for model moves. This would lead to higher cost of the alignment in Figure 10(b), than of the alignment in Figure 10(c), thereby excluding the former from the resulting set of optimal alignments. Depending on the domain, one can decide if it is more likely that an activity is erroneously missing or added to a log.

*Enhancement of Process Models*

The last category in process mining covers problems that take an existing process model and an event log as input and enrich the models with information

contained in the event logs. Enhancement techniques can visualize bottlenecks, service levels, throughput times, and frequencies [4].

A particular area of research in the enhancement category is *model repair*[66]. It might occur that a process model does not perfectly fit to the event log, because for instance some exceptional situation was not considered in the course of creating the model, but happens in reality and is recorded. Then, it is possible to automatically adapt business process models to better reflect observed event logs with the techniques proposed in [66, 46]

We briefly summarize this section about process mining. We introduced *event logs*, cf. Definition 5, explained the notion of alignments, cf. Section 2.3, which can be used for conformance checking, and mentioned the three main types of process mining, i.e., discovery, conformance, and enhancement. In this thesis, we will not address discovery, but instead assume that models exist already—either by manual creation or automatic discovery.

## 2.4 MONITORING ARCHITECTURES

In the previous section, we introduced event logs, which we require in the remainder of the thesis. In many business process execution environments (e.g., in healthcare [117]), however, there is no central workflow engine that orchestrates the process execution and generates event logs.

For such cases, researchers have developed monitoring architectures that provide basic monitoring capabilities using heterogeneous information sources [96, 198, 53, 88]. We shall briefly review these frameworks and architectures in the following.

The assumption of the investigated monitoring architectures is that traces of process execution are scattered in the IT landscape of an organization. Various systems and tools are used during process execution for specialized purposes, e.g., customer relationship management tools, enterprise resource planning systems. Therefore, one requirement for monitoring architectures is to provide event collection and correlation mechanisms for events from different sources. If an organization has elicited process models that capture its business processes, these models can serve as overview and monitoring indicator for single instances or as aggregated view on a set of instances. For example, the currently running activity (or activities, if the process contains parallelism) can be visualized in the model for each process instance. Further, the already terminated activities that led to the current activity can be highlighted to visualize the execution path of a certain process instance.

Figure 11 shows an example monitoring framework that performs event collection, event aggregation, and event correlation with process instances. The figure shows the components that are required to connect the heterogeneous raw event sources (e.g., spreadsheets, information systems). In this framework, specific event monitoring points (EMPs) are defined, with which events can be correlated. These EMPs attach to process models that are representable as a graph. We adopt our definition in [88].

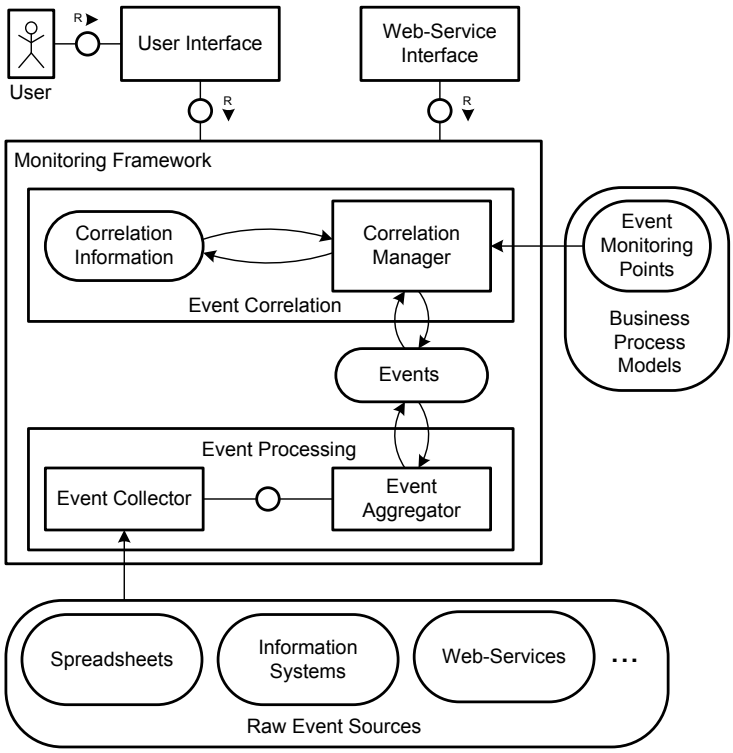



Figure 11: Example monitoring framework. The figure depicts an FMC block diagram [111]. At the bottom, different sources for raw events are shown which are collected and aggregated by the event processing component. Then, the correlation component assigns the aggregated events to so called *event monitoring points* in the business processes. We assume that such an architecture is used and that correlated events can be transformed into event logs.

**Definition 6 (Event Monitoring Point)** *Let GN be a set of nodes in a procedural process model. Let further LCT be the set of possible transitions in the lifecycle of a process model element (e.g., start, or end of an activity). Then, an event monitoring point (EMP) is a tuple* $\mathfrak{m} = (gn, lct)$*, that is, a state transition* $lct \in LCT$ *within the lifecycle of a node* $gn \in GN$*.*

Event monitoring points mark possible candidates in process models, with which events of the surrounding IT infrastructure can be correlated. For example, a new patient entry in a hospital information system might indicate the termination of the activity *enter patient into system* of a certain clinical process. In this thesis, we will return to the notion of event monitoring points (EMPs), when investigating the question, which of these EMPs should be used for monitoring in Chapter 6.

Beside the work in [88], which presented a formal and rather abstract definition of the problem of monitoring in non-automated process execution environments, there are several related approaches, which we shall discuss briefly in the following.

An ontology for business process analysis was presented by Pedrinaci et al. in [156]. They distinguish between process monitoring events and activity monitoring events, and present a detailed activity lifecycle, cf. [206], which they name

"events ontology state model". We do not want to predefine a specific ontology, but we abstract from concrete lifecycles of activities and events in Definition 6.

A research branch deals with event driven business process management, cf. [23]. The sixth workshop [179] on this topic was held in the year 2012. For example, using complex event processing to listen to and tap into different streams of events, Janiesch et al. [96] combine complex event processing and business process management methods to enable monitoring of distributed business processes. In their work, the focus is on the integration of several organizations involved in one process. They use an event processing network of specialized agents that perform, for instance, filtering, transformation, or pattern detection.

Another approach for correlating events to processes is based on semantic ontologies introduced by Thomas et al. [198]. Their approach relies on agents that use ontologies to perform the monitoring of process instances. This enables a flexible configuration of monitoring. Similarly, semantic technologies for monitoring are proposed by Costello and Molloy [53]. In latter work, semantic rules can be specified to capture exceptions, and metrics for the performance of processes are calculated by linking events to processes. A method that works directly on the data to identify potential candidate correlation rules is described by Rozsnyai et al. in [185]. Their approach computes statistics such as uniqueness of values for data attributes to identify candidates for correlation. Based on the statistics, their algorithm finds correlation pairs and calculates confidence values based on the statistics. In this thesis, we assume that correlation of events to process instances is possible and the $\beta$-function assigning each event in the event log to a case is given.

Although there exist several approaches to allow monitoring in heterogeneous environments, the research on bridging the gaps between the low level events and the business process models ongoing. Herzberg et al. [89] recently proposed a framework for monitoring events in a distributed setting. The framework is flexible and allows the definition of bindings between events and process activities on multiple layers of abstraction.

## 2.5 PROBABILISTIC MODELS

Recall the motivating setting in Chapter 1, that is, for some activities in business processes we do not know the exact occurrence of their events. In this setting, we want to calculate probabilistic estimations for the affected events. Therefore, we recall basics of probability theory in this section. First, let us discuss random variables and probability distributions, as these are used to capture time in our model.

### *Random Variables and Probability Distributions*

This thesis is not primarily about the mathematical details of probability, and the interested reader is referred to the classical work by Feller [68], or the textbook by Billingsley [36]. We mainly require probability distributions in the continuous domain.

In general, we look at *sample spaces* $\mathbf{S}$, which capture the possible outcomes of an experiment. The family of subsets of $\mathbf{S}$ is denoted as $\mathcal{F}$, such that for each event $E \in \mathcal{F}$ a probability value of occurrence can be assigned. That is, $P(E)$ is interpreted as the probability that the event $E$ occurs. We adopt the definition of random variables from Trivedi [199, Chapter 3]:

**Definition 7 (Random Variable)** *A random variable $X$ on a probability space $(\mathbf{S}, \mathcal{F}, P)$ is a function $X : \mathbf{S} \to \mathbb{R}$ that assigns a real number $X(s)$ to each sample point $s \in \mathbf{S}$, such that, for every real number $x$, the set of sample points $\{s \mid X(s) \leqslant x\}$ is an event, that is, a member of $\mathcal{F}$.*

For this work, we use random variables to model duration distributions, that is, we are interested in the probability that an activity is completed at a certain point in time. Therefore, we consider probability distribution functions of random variables.

**Definition 8 (Probability Distribution Function)** *The (cumulative) distribution function $F_X$ of a random variable $X$ is defined as*

$$F_X(x) = P(X \leqslant x), \quad -\infty < x < \infty$$

In the above definition, we use the subscript $X$ to indicate the random variable described by the distribution function. We omit the subscript notation and write $F(x)$, instead of $F_X(x)$, when it is clear, to which random variable we refer with $F(x)$. A probability distribution function yields probability values, i.e., the values are between $0$ and $1$. Further, because the cumulative nature of a probability distribution, the function in monotonically increasing and has the limits $\lim_{x \to -\infty} F(x) = 0$ and $\lim_{x \to \infty} F(x) = 1$.

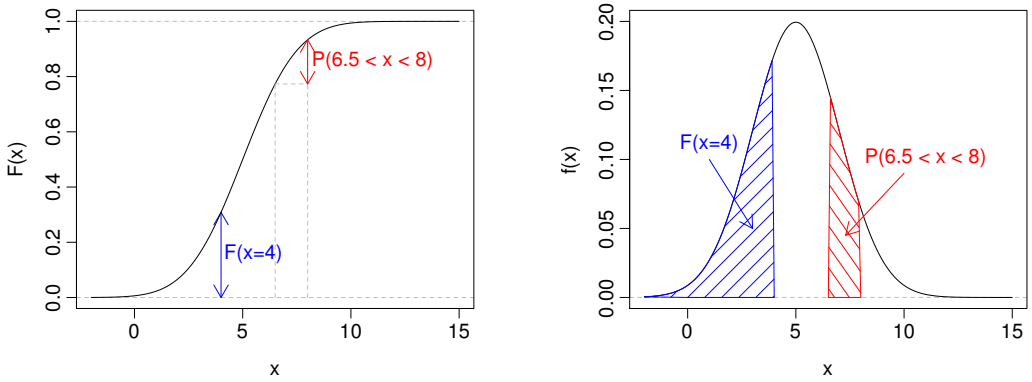
In this thesis, we consider *continuous* random variables, because time is continuous, as well. That is, the distribution functions $F(x)$ that we use in this thesis are continuous in the entire domain of $x \in (-\infty, \infty)$. That is, the derivative of the distribution function $dF(x)/dx$ exists everywhere (except at perhaps a finite number of points). For continuous random variables, we are also interested in its the density function.

**Definition 9 (Probability Density Function)** *The probability density function of a continuous random variable $X$ is defined as $f_X(x) = dF(x)/dx$*

We use the terms *probability density function* and *density function* interchangeably. Note that density functions are not reflecting probabilities, i.e., the density function can have values higher than 1. Of particular interest is the normal distribution that has the following density function.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Thus, the normal distribution is a parametric model for a probability density, and has two parameters, a mean $\mu$ and a standard deviation $\sigma$. The mean marks the center point around which the values are distributed in a bell-shaped curve,




(a) probability distribution function.

(b) probability density function.

Figure 12: Relation of distribution function and density function. The probability distribution function (a) corresponds to the integral of (resp. the area under) the density function (b).

while the standard deviation $\sigma$ captures the spread of the values around the mean. For convenience, we denote the normal distribution with $\mathcal{N}(\mu, \sigma^2)$, that is the mean and the variance $\sigma^2$.

In Figure 12, the normal distribution $\mathcal{N}(\mu = 5, \sigma^2 = 4)$ is depicted. The *distribution function* is captured in Figure 12a. We can see that the values are in the range of $0 \leqslant F(x) \leqslant 1$, which is a requirement for probabilities. For example, the value of the distribution function $F(x)$ at $x = 4$ is around 0.308, which means that the probability that the depicted random variable takes on values less or equal to 4 with a probability of 30.8 percent. We get the same result, if we look at the integral of the density function $\int_{-\infty}^{4} f(x)dx$, which corresponds to the shaded part on the left side in Figure 12b.

We can also derive the probability of an outcome of $X$ between two values, that is, the probability that an outcome lies between $a$ and $b$ is $P(a < X < b) = F_X(b) - F_X(a)$ for all $a \leqslant b$. An example for $a = 6.5$ and $b = 8$ is depicted in Figure 12a as the difference between two values, and in Figure 12b as the area under the density curve between the values 6.5 and 8.

Often, when we reason about a random variable, we are interested in the *expected value* of the random variable. In our setting, we want to know the expected duration of activities, which can be used as basis for predicting of remaining durations.

**Definition 10 (Expected Value)** *Let $X$ be a continuous random variable with density function $f$. Then, the expected value (or mean) $E(X)$ of $X$ is given by*

$$E(X) = \int_{-\infty}^{\infty} x f(x) dx.$$

The expected value $E(X)$ is also the *first moment* of a random variable X. More generally, moments characterize a function, and one is often also interested in

additional moments of a density function. We define moments about the mean, or central moments as follows.

**Definition 11 (Central Moments)** *Let X be a continuous random variable with mean* $\mu_X = E(X)$. *Then we define the* $n$*th central moment* $\mathbf{M}_n$ *as*

$$E\left((X-\mu_X)^n\right) = \int_{-\infty}^{\infty} (x-\mu_X)^n f(x)dx.$$

The second central moment is called the *variance* of a probability distribution. This quantity expresses how wide the outcomes of a random variable spread around the mean. That is, there can be two different random variables with the same mean (or expected value), but they can have significantly different variances.

Further moments can be of interest, e.g., the third moment, which when standardized is called the *skewness*. Standardization of the $n$th moment is achieved by dividing the moment by $\sigma^n$. In this case, the skewness is defined as follows.

**Definition 12 (Skewness)** *Let X be a continuous random variable. Its skewness is defined as its standardized third central moment:*

$$E\left(\left(\frac{X-\mu_X}{\sigma_X}\right)^3\right) = \frac{E\left((X-\mu_X)^3\right)}{E\left((X-\mu_X)^2\right)^{\frac{3}{2}}}$$

We introduced skewness, because it is an established way to describe how the values of a random variable are distributed around the mean. For example, a random variable with positive skew (i.e., a right-skewed variable) describes a distribution of values that is more compact to the left of the expected value and more widespread to the right. We encounter this phenomenon in many real processes, where most of the cases are finished in a certain time span, but some outliers that take longer exist and need to be considered. The log-normal distribution, for example, has a positive skewness, and can be used to model such phenomena [196].

We want to provide the following analogy that might help to get a better intuition of random variables and their properties. The analogy is targeted to readers who have tried to balance a seesaw on a playground. Imagine a very long board, on which we strew sand according to a certain distribution. When more sand accumulates at a certain position, a heap is formed. The *expected value* (or mean) of the distribution corresponds to the center of mass of the sand on the board, where we can balance the board (assuming that the board itself is weightless). The *variance* reflects how far the sand scatters from the center of mass on average. Finally, if the distribution is *right-skewed*, more sand lies to the left of the center of mass than to the right. But the sand on the left is closer to the center of mass and has a shorter level, while the sand on the right is farther away and has a longer level, such that balance is maintained.

Next, we continue with dependencies between random variables.

*Conditional Probability and Bayes' Rule*

Let X, Y be random events with probability of X not zero ($P(X) \neq 0$). Then, the formula to calculate the conditional probability of an event Y, given the occurrence of event X is:

$$P(Y \mid X) = \frac{P(X \cap Y)}{P(X)} \tag{2}$$

Note that if the events are *independent*, it holds that $P(Y \mid X) = P(Y)$, that is, $P(X \cap Y) = P(X)P(Y)$. Equation 2 should be read as "the conditional probability of event Y, given that event X has taken place." The equation can be transformed into the *chain rule* of conditional probabilities:

$$P(X \cap Y) = P(X)P(Y \mid X) \tag{3}$$

More generally, if $X_1, \ldots, X_n$ are events, then we can write:

$$P(X_1 \cap \ldots \cap X_n) = P(X_1)P(X_2 \mid X_1) \cdots P(X_n \mid X_1 \cap \ldots \cap X_{n-1}) \tag{4}$$

The chain rule is useful, as it allows us to express the probability of a combination of events as the probability of the first, the probability of the second given the first, and so on.

After manipulating Equation 2, we arrive at *Bayes' Rule* that allows us to reformulate a conditional probability $P(X \mid Y)$ using its inverse $P(Y \mid X)$:

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{P(Y)} \tag{5}$$

An example of reasoning with conditional probabilities shall make these formulae clearer. Therefore, we adopt the example from [169]. Suppose, we have two dice. One is a fair die with $\frac{1}{6} \approx 0.166$ probability to roll a 6 and a loaded die that has a 0.6 chance of rolling a 6. Further, suppose we do not know which die is which one and pick a die randomly. Thus, we know that

$$P(\text{fair}) = 0.50 \quad \text{and that} \quad P(\text{loaded}) = 0.50.$$

We are interested in finding out which die we picked by rolling it. We also know the conditional probabilities:

$$P(6 \mid \text{fair}) = 0.166 \quad \text{and} \quad P(6 \mid \text{loaded}) = 0.6$$

We can compute the probabilities of the joint events with Equation 3.

$$\begin{aligned} P(6 \cap \text{fair}) &= P(6 \mid \text{fair})P(\text{fair}) \\ &= 0.166 \cdot 0.5 = 0.083 \\ P(6 \cap \text{loaded}) &= P(6 \mid \text{loaded})P(\text{loaded}) \\ &= 0.6 \cdot 0.5 = 0.3 \end{aligned}$$

So far, we only stated the probabilities for this example. We are more interested in finding out which die is the loaded one. Therefore, we roll a randomly picked

die. What can we infer from the observed outcome? Suppose, we roll a 6. We can achieve this result with either of the two dice. Thus, the event has the probability

$$P(6) = P(6 \cap \text{fair}) \cup P(6 \cap \text{loaded}) = 0.083 + 0.3 = 0.383$$

We can insert these values into Bayes' Rule in Equation 5 and derive the probability that the dice we picked randomly is loaded, given the evidence of an observed 6 at the first roll.

$$P(\text{loaded} \mid 6) = \frac{P(6 \cap \text{loaded})}{P(6)} = \frac{0.3}{0.383} = 0.78$$

Note how the probability that we picked the loaded die changed from 0.5 (without knowledge) to 0.78 (with one observation of a randomly rolled 6). In Bayesian statistics, the former probability is called *prior*, and the latter probability is called *posterior*. The computation of posterior probabilities given prior probabilities and observations is called *inference*.

Suppose, we throw the die a second time, and we get another 6. Then, the probability that we picked the loaded die is updated to:

$$P(\text{loaded} \mid 6,6) = \frac{P(6,6 \cap \text{loaded})}{P(6,6)} = \frac{0.6^2 \cdot 0.5}{0.6^2 \cdot 0.5 + 0.166^2 \cdot 0.5} = 0.928$$

Notice how the probability of holding the loaded die increases to approximately 93 percent, when observing two rolls with the number 6, consecutively. In contrast, the probability that the picked die is the fair one (the complementary event) is reduced to $1 - 0.928$, that is around 7 percent, only. We will never have absolute certainty, but we can increase our confidence in our belief, by increasing the number of dice throws.

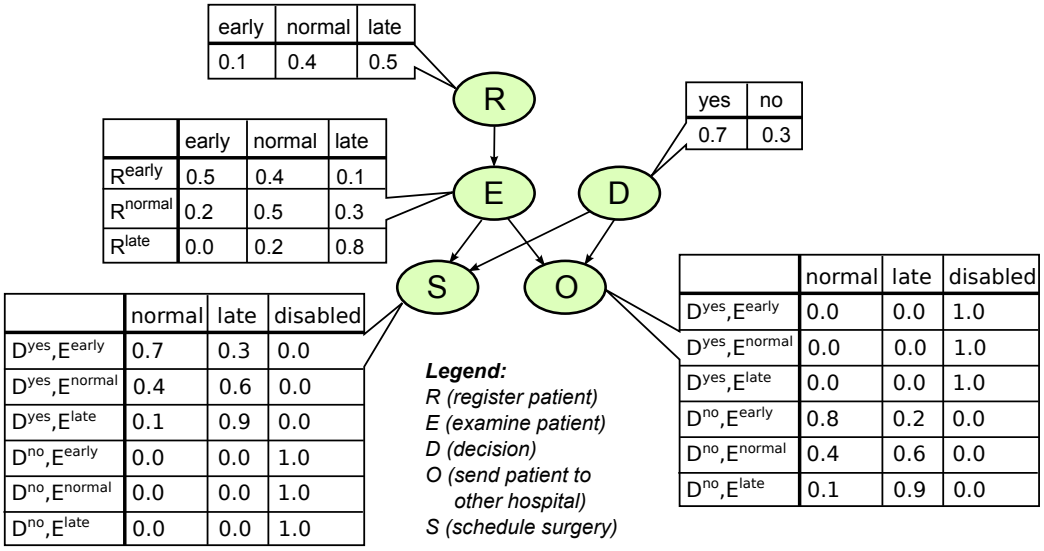
In fact, this set of equations (i.e., Equations 2–5) allows us to compute probabilities in more complex settings as well. Once we have dependencies between multiple random variables, however, we can resort to graphical representations to visualize these for better understanding.

### *Graphical Probabilistic Models*

There exist several forms of graphical probabilistic models [102], e.g., Bayesian networks [154], neural networks, factor graphs, Markov random fields, Ising models, and conditional random fields. These models serve multiple purposes and have different properties. They share an underlying probabilistic model consisting of a set of random variables with interdependencies, but usually differ in structural properties as well as expressiveness.

In this thesis, we will use Bayesian networks, which have application areas in, for example, stochastic modeling, machine learning, classification, text recognition [113]. One advantage of using Bayesian networks, as opposed to the other models, is that we can naturally encode prior knowledge in the model as prior probabilities, and update these based on new observations.

Bayesian networks model a set of random variables that are in relation to each other, i.e., random variables can be dependent or independent from each other.




| early | normal | late |
|---|---|---|
| 0.1 | 0.4 | 0.5 |

| | early | normal | late |
|---|---|---|---|
| $R^{early}$ | 0.5 | 0.4 | 0.1 |
| $R^{normal}$ | 0.2 | 0.5 | 0.3 |
| $R^{late}$ | 0.0 | 0.2 | 0.8 |

| yes | no |
|---|---|
| 0.7 | 0.3 |

| | normal | late | disabled |
|---|---|---|---|
| $D^{yes},E^{early}$ | 0.7 | 0.3 | 0.0 |
| $D^{yes},E^{normal}$ | 0.4 | 0.6 | 0.0 |
| $D^{yes},E^{late}$ | 0.1 | 0.9 | 0.0 |
| $D^{no},E^{early}$ | 0.0 | 0.0 | 1.0 |
| $D^{no},E^{normal}$ | 0.0 | 0.0 | 1.0 |
| $D^{no},E^{late}$ | 0.0 | 0.0 | 1.0 |

| | normal | late | disabled |
|---|---|---|---|
| $D^{yes},E^{early}$ | 0.0 | 0.0 | 1.0 |
| $D^{yes},E^{normal}$ | 0.0 | 0.0 | 1.0 |
| $D^{yes},E^{late}$ | 0.0 | 0.0 | 1.0 |
| $D^{no},E^{early}$ | 0.8 | 0.2 | 0.0 |
| $D^{no},E^{normal}$ | 0.4 | 0.6 | 0.0 |
| $D^{no},E^{late}$ | 0.1 | 0.9 | 0.0 |

Figure 13: Example Bayesian network. A Bayesian network with discrete variables corresponding to the middle part of the model in Figure 3 is depicted with conditional probability distributions. The variables of the activities indicate the occurrence of their termination respectively (e.g., variable $E$ can have values *early*, *normal*, and *late*). The conditional probability distributions are given conditioned on the values of the parents, such that each row sums to 1.

The formalism was presented by Pearl in [153]. It captures random variables in a directed acyclic graph, where the nodes capture random variables, and the edges represent conditional dependencies between random variables they connect.

**Definition 13 (Bayesian Network)** *Let $\{X_1, \ldots, X_k\}$ be a set of random variables. A Bayesian network BN is a directed acyclic graph $(N, F)$, where*

- $N = \{n_1, \ldots, n_k\}$ *is the set of nodes assigned each to a random variable $X_1, \ldots, X_k$*
- $F \subset N \times N$ *is the set of directed edges.*

*Let $(n_i, n_j) \in F$ be an edge from parent node $n_i$ to child node $n_j$. The edge reflects a conditional dependency between the corresponding random variables $X_i$ and $X_j$.*

*Each random variable is independent from its predecessors given the values of its parents. Let $\pi_i$ denote the set of parents of $X_i$. A Bayesian network is fully defined by the probability distributions of the nodes $n_i$ as $P(X_i \mid \pi_i)$ and the conditional dependency relations encoded in the graph. Then, the joint probability distribution of the whole network factorizes according to the chain rule as $P(X_1, \ldots, X_N) = \prod_{i=1}^{N} P(X_i \mid \pi_i)$*

Figure 13 shows an example Bayesian network for the surgery model that we introduced in Figure 3 on page 12. The nodes qualitatively represent the activity terminations—that is, we discretize time in this example and use values such as *early*, *normal*, or *late*. The root nodes are the *register patient* activity denoted by the variable $R$, and the *decision* (captured in variable $D$) to perform the surgery (value *yes*), or send the patient to another hospital (value *no*). Note that by this choice of modeling, the decision does not depend on the time of the examination, or the other activities' time. The specified conditional probability distributions are given as tables. Here, for all combinations of the values of the parent variables,

we need to specify the probabilities of the elements of the variable given the combination. That is, we need to specify the rows in these tables as probability distributions that sum to 1. Consider the second row of the conditional probability table of variable S in Figure 13. Here, we see that the schedule surgery activity, given the parent values of $D = yes$ and $E = normal$, is assigned the *normal* state with probability 0.4, and the *late* state with probability 0.6. The probability of the *disabled* state, given that the decision is *yes* is 0. These conditional probabilities fully specify the probability distribution of the joint network and allow us to reason about *marginal* probabilities (e.g., we can compute the probability $P(S = late) = 0.483$ of the schedule surgery activity to be late).

Further, if we obtain knowledge about a certain case, for example, if we see that the registration was done late, and the decision for the surgery is positive, we can condition the model on this observation and compute the posterior probability of the surgery to be delayed as $P(S = late \mid R = late, D = yes) = 0.78$. It is possible to also reason backwards in the Bayesian network. For example, if we only observe that the schedule surgery activity has been completed normally (i.e., $S = normal$), we can compute the probability of the registration being late as $P(R = late \mid S = normal) = 0.355$—that is, the probability that the surgery started late in this case decreased from 50 percent to 35.5 percent.

There has been a lot of research on Bayesian networks, cf. [113]. Topics include analysis [73], or representation, where different flavors of Bayesian networks were proposed, e.g., dynamic Bayesian networks allow the structure of the model to change over time [142]. Qualitative probabilistic networks [205, 63] abstract from numeric probability values and only consider effects of variables on each other. Pfeffer [160] introduces ProPL, a probabilistic programming language that is translated to a dynamic Bayesian network. The language describes processes that evolve over time and consist of sub-processes that can have random durations.

Besides representation, a challenging research problem is *inference* in Bayesian networks, which is proved to be NP-hard [52], except for the special case that the underlying network structure is singly connected (also called *polytree*), for which reasoning is linear in the size of nodes. The complexity of inference spawned a lot of heuristics to approach the problem. One heuristic is loopy belief propagation, which iteratively sends messages of current beliefs in the network until convergence [55]. An approach that passes Gaussian beliefs to neighboring nodes can be found in the work by Minka [135]. We can but scratch the surface of the topic of inference in this thesis, and refer to the textbook by Koller and Friedman [113], as well as to the overview on inference techniques by Darwiche in [54].

There exists a special case of Bayesian networks, where every random variable has a normal distribution and its mean can be specified by a linear combination of the states of the parent nodes.

**Definition 14 (Linear-Gaussian model)** *A linear Gaussian model is a Bayesian network, where all random variables are normally distributed and can be expressed by a*

*linear combination of their parents. That is, for each random variable* $X_i$ *in the net, we can express the conditional probability density function given the parents* $\pi_i$ *as [38]:*

$$P(X_i \mid \pi_i) = \mathcal{N}\left(\sum_{j \in \pi_i} w_{ij} x_j + b_i,\, v_i\right)$$

*where* $w_{ij}$ *and* $b_i$ *are parameters that capture the mean based on the value* $x_j$ *of the respective parent, and* $v_i$ *is the variance of the conditional distribution for* $X_i$

Linear Gaussian models are of special interest, because we can perform inference in this type of Bayesian networks in $\mathcal{O}(n^3)$ [113], where $n$ is the number of nodes.

Having introduced this set of fundamental models and tools, we can focus on the main contributions of this thesis in the following part.

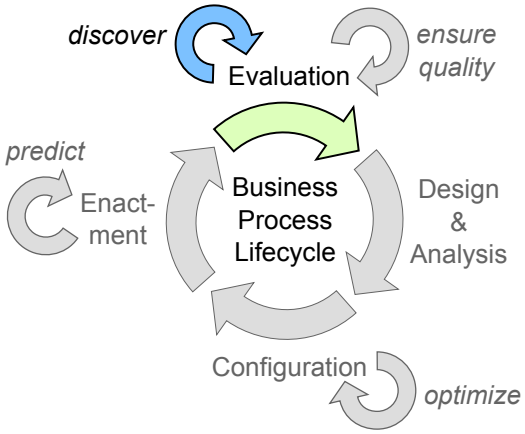
Part II

# PROBABILISTIC ESTIMATION OF UNOBSERVED PROCESS EVENTS

# 3

# DISCOVERY OF STOCHASTIC PETRI NET MODELS

*This chapter is based on the results published in [171].*




This chapter deals with obtaining the performance model of business processes that is used in the remainder of the thesis. In terms of the business process lifecycle, this chapter belongs to the *evaluation* phase. In this phase, we want to analyze the durations of individual activities and of decision probabilities based on the data that we get from an enacted business process. Recall that we use generally distributed transition stochastic Petri net (GDT_SPN) models, cf. Definition 4 on page 16, as our formalism, which extend GSPNs by allowing arbitrary delay distributions for transitions.

## CHAPTER OUTLINE

First, in Section 3.1, we present related approaches that also obtain performance models. Then, in Section 3.2, we discuss preliminary concepts and a solution idea. In Section 3.3, we outline the challenges that are inherent to the problem. Our approach to deal with the challenges is presented in Section 3.4. Section 3.5 deals with the evaluation of how accurately we can discover model performance parameters in theory. Finally, we discuss the evaluation findings in Section 3.6.

## INTRODUCTION

Obtaining accurate performance models of business processes is important for any organization, as these models allow for prediction, simulation, analysis, and improvement of business processes. Our goal is to find the stochastic model that best explains the behavior recorded in an event log. Therefore, we need to carefully examine the event log to identify how long the activities took in reality and how frequent different possible paths in the model are.

Strictly speaking in the terminology of process mining, cf. Section 2.3, we present a method of *enhancement*, but we want to *discover* the performance characteristics. More explicitly, we take a model and enrich it with performance information stored in an event log. Recall that event logs in this work are a common interface to different information sources (e.g., hospital information system, workflow management systems, or spreadsheet data) and thereby abstract from specific data formats.

## 3.1 RELATED WORK

Estimation of model parameters based on real observations helps to gain insights into the performance of a system. An extensive survey of performance evaluation methods is provided by Heidelberger and Lavenberg in [86]. Rolia and Vetland present statistical methods to obtain parameters of a queueing model for distributed systems in [178].

In the context of workflow systems, van der Aalst et al. describe how annotated transition systems can be enriched with performance information and successively be used for prediction of remaining durations [10]. One limitation of their approach, however, is the inability to handle parallelism correctly. Our approach is based on the more expressive Petri net formalism, which does not have this limitation. Related work by Nakatumba and van der Aalst, that also looks at the performance encountered in event logs, investigates the effect of resource work load on service times in workflow executions [147].

There already exists research on obtaining Petri net models with stochastic performance characteristics from data. Hu et al. propose a method to mine exponentially distributed SPN models from workflow logs in [91] considering the firing rates of transitions. Another approach was proposed by Anastasiou et al. [24] and uses location data to elicit generalized stochastic Petri net (GSPN) [130] models for modeling flows of customers. They fit hyper-erlang distributions to transition durations representing waiting and service times and replace the corresponding transitions with a GSPN subnet exhibiting the same characteristics of the hyper-erlang distribution. Their approach considers every transition in isolation though, which poses no problems in serial processes. Parallelism in the processes, especially multiple parallel transitions in conflict, are not covered in that approach. Similarly, Janeček et al. propose to use stochastic Petri nets with the Coxian distribution (a variant of phase-type distributions) in [95]. While the authors argue that these models lend themselves for analytic methods of performance analysis, they do not discuss execution policies of these models.

There are also attempts at eliciting non-Markovian stochastic Petri nets from data. Leclercq et al. investigate how to extract models of normally distributed data in [115]. Their work is based on an expectation maximization algorithm that they run until convergence. In comparison to our approach, they are not able to deal with missing data and do not consider different execution policies.

Reconstructing model parameters for stochastic systems also has been investigated by Buchholz et al. in [45]. They address the problem to find fixed model parameters of a partially observable underlying stochastic process. In contrast to our work, the underlying process's transition duration distributions need to be specified beforehand, while our aim is to infer also transition duration distributions of a GDT_SPN model. In a similar setting, i.e., with incomplete information, Wombacher and Iacob estimate duration distributions of activities and missing starting times of processes in [212]. The authors make the assumption that work is interrupted by other undocumented work. The difference to our approach is that they try to filter duration distributions of activities to remove external effects. We do not filter the duration of activities, but instead assume that external conditions are in a steady state (i.e., the impact on the work performance is not changing). We assume that we improve the overall model quality by implicitly capturing external conditions in the model.

In [183], Rozinat et al. investigate how to gather information for simulation models. They try to identify data dependencies for decisions and mean durations and standard deviations. Thereby, they do manual replay, which is not necessarily finding an optimal alignment between model and log. The approach that we propose is capable to deal with noise in a more robust way, by building on the notion of alignments [15, 5], which identifies an optimal path through the model for a noisy trace. Similarly to Rozinat, Măruşter and van Beest mine heuristic nets from event logs and convert them in a later step to Petri net models in [131]. They do not provide an algorithm, but perform these steps by hand. Execution policies that are required for stochastic Petri nets with arbitrary delay distributions are excluded from their discussion, as their focus is on redesigning the process.

## 3.2 CONCEPTIONAL OVERVIEW

To enrich a Petri net model with stochastic information gathered from historical observations, we need a Petri net model and an event log. Figure 14 provides an overview of the elicitation approach, as proposed in [171]. The inputs to the approach are a Petri net *model* reflecting the structural behavior of the process, and an *event log* containing the traces that represents actual executed process cases. Further, a *configuration* is necessary, as GDT_SPN models are flexible in their execution policies [128] and transition distribution types (e.g., normal, uniform, exponential, non-parametric) that will be fit to the extracted transition durations.

We propose a pre-processing step to ensure a good *fitness* between model and event log, see Section 2.3. Therefore, we compute the fitness as described in Equation 1 on page 20, and if the fitness value is lower than a user defined threshold (e.g., 0.7), we propose to first apply techniques to repair the model to

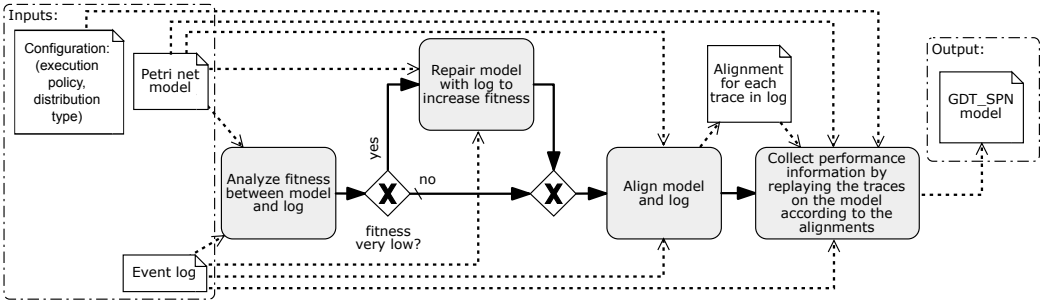



Figure 14: Elicitation process overview. This model describes the conceptual method to enrich a Petri net model with stochastic execution information gathered from event logs. We also presented this figure in [171].

capture the behavior in the event log. For this repair step, we rely on the work by Fahland and van der Aalst [66], which adds subprocesses to the model to allow for deviations observed in the log. Alternatively, we can also use the method proposed by Buijs et al. in [46] that offers additional flexibility and allows the user to set desired parameters for the resulting model in terms of fitness, but also supports other quality metrics, i.e., precision, simplicity, generalization, and structural similarity to the original model. Latter work uses a genetic algorithm approach to find candidate models. Beside these automatic approaches, the repair of the model can also be done by experts manually.

Once we made sure that the event log and the model can be aligned resulting in a reasonable fitness score, we can turn to the actual enrichment of the model. Therefore, we use the notion of alignments [15], as we introduced in Section 2.3, and select an optimal alignment for each trace in the event log. It is not guaranteed that there is only one optimal alignment. Especially, if the model contains parallelism, the order of missing events is not defined by the model, and different orderings of parallel events do not affect the costs. In the current approach, we let the alignment approach decide non-deterministically between multiple alignments that have the same costs. Note that if the trace fits the model perfectly, the alignment is usually deterministic. An exception to latter rule is for example that two transitions with equal labels are executed in parallel. Then, it is not clear, to which of the parallel activities the two corresponding events in the trace are aligned.

$$\begin{aligned} tr_1 : &\quad \langle\, A(0.0), \quad B(1.6), \quad D(1.9), \quad C(4.2), \quad B(4.46) \,\rangle \\ tr_2 : &\quad \langle\, B(0.0), \quad D(5.81), \quad D(7.0) \,\rangle \end{aligned}$$

Figure 15: Example event log with time. We extend the event log from Figure 8 with time annotations for the following discussion of the enrichment approach.

The selected alignment for a trace defines a path in the model. Let us revisit the event log introduced in Section 2.3. But this time, we also consider timing information. Let $e \in E$ be an event, and $\alpha(e) = A$ We use the shorthand notation of $A(1.3)$ to denote an event $e$ that happened at time 1.3, i.e., $\gamma(A) = 1.3$ and belongs to activity $A$. Figure 15 is based on the example introduced in Figure 8, annotated with timestamps. By convention, each trace starts at time 0, as for our

purposes we are only interested in relative durations after the start of the case. Reconsider trace $tr_1$ with the following alignment to the model in Figure 7

| *log* | A(0.0) | ≫ | B(1.6) | D(1.9) | ≫ | ≫ | ≫ | C(4.2) | B(4.46) | ≫ | ≫ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *model* | A | $\tau$ | B | D | $\tau$ | $\tau$ | $\tau$ | C | B | $\tau$ | $\tau$ |
| | $\mathbf{t}_A$ | $\mathbf{t}_1$ | $\mathbf{t}_B$ | $\mathbf{t}_D$ | $\mathbf{t}_2$ | $\mathbf{t}_4$ | $\mathbf{t}_1$ | $\mathbf{t}_C$ | $\mathbf{t}_B$ | $\mathbf{t}_2$ | $\mathbf{t}_3$ |

This alignment yields a path in the Petri net model of Figure 7, as depicted in Figure 16. Here, we projected the alignment for trace $tr_1$ into the model. The black part marks the path that the trace takes in the model according to the alignment. Once we determined the path through the model, we can begin with the collection of the stochastic execution information.

*Alignment Based Replay*

In essence, the enrichment algorithm proceeds as follows. We replay the trace according to the alignment in the model. Replay is done using a global clock, which starts at zero with the first event in the trace. The global clock is updated with each transition firing that reflects the occurrence of events in the trace (i.e., with each synchronous move). Each transition is equipped with its own timer that counts the enabling time of that transition. Once a transition becomes enabled during replay, we remember the current point in time, which we get from the global clock. When the transition fires, we compute the firing delay as the difference between the global clock's timestamp and the remembered local enabling timestamp. If the transition fires synchronously with the log, the global clock time is updated first before latter computation. Special care has to be taken with immediate transitions, as well as different execution policies [128], which we will discuss later.

Beside temporal information, we are interested in the probabilities of decisions. For example, in the model in Figure 7 on page 20, we want to discover the relative weights of transition $\mathbf{t}_3$ and $\mathbf{t}_4$. These transition weights represent the probability to leave the cycle $\mathcal{W}(\mathbf{t}_3)$, and the probability to remain in it, $\mathcal{W}(\mathbf{t}_4)$, respectively. Therefore, we need to count the number of times a transition fired in each *marking*[1]. To determine the weights of $\mathbf{t}_3$ and $\mathbf{t}_4$, we would consider marking $\{p_7\}$, i.e., the marking, in which only place $p_7$ contains a token. The ratio of the firing counts per marking captures the observed ratio of the firings which happened in a particular marking.

Let us assume that in 74 cases out of 129, the next transition to fire in marking $\{p_7\}$, was transition $\mathbf{t}_3$, which leaves the cycle. In this case, the best estimate for the weight of $\mathbf{t}_3$ would be $\frac{74}{129}$, and respectively $\frac{55}{129}$ for $\mathbf{t}_4$. The stochastic confidence in these fractions rises with the number of observations, i.e., we are more confident that the real probability of picking transition $\mathbf{t}_3$ in marking $\{p_7\}$ is $\frac{74}{129}$, if we observed that it was chosen in 740 out of 1 290 cases, than in the example before.

1 A marking of a Petri net is a function that assigns each place a number of tokens. We use a shorthand set notation for places that contain one token, see Section 2.2

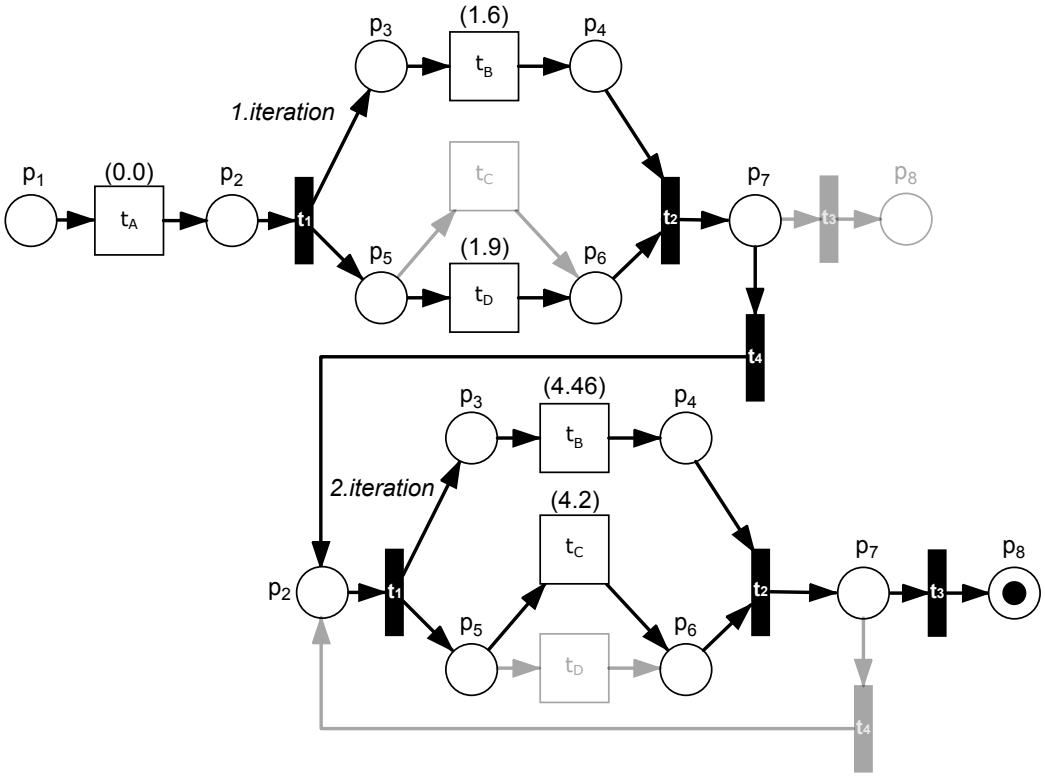



Figure 16: Alignment in the model. The alignment of trace $tr_1$ is shown in the model. We depicted the second iteration separately, because the alignment traverses the cycle twice. Black parts mark the path that was taken in the alignment; the gray parts represent omitted choices. We annotate the timing of the events of $tr_1$ at the top of the transitions, e.g., the event of transition $\mathbf{t}_D$ occurred at time 1.9.

## 3.3 CHALLENGES

Certain difficulties arise when trying to extract stochastic execution information from event logs. These are not obvious, however, and we therefore explain them in greater detail, before we turn to the algorithm.

### *Multiple Execution Policies*

As we mentioned before in Sect. 2.2, there exist multiple execution policies for GDT_SPN models [128]. This is due to the fact that we allow to use other distributions than the negative exponential distribution for timed transitions.

In order to determine the execution policy, decisions have to be taken with regard to the selection of the next transition, and the memory properties of enabled transitions that did not fire. The discussion by Marsan in [128] offers more details on the different possible execution policies for stochastic Petri nets. In this thesis, we consider the following selection policies:

GLOBAL PRESELECTION In this selection policy, the next timed transition is randomly chosen from all currently enabled timed transitions based on their

respective weights. Subsequently, the duration of the chosen transition is determined by drawing a random sample from its duration distribution.

RACE-POLICY In this selection policy, all enabled timed transitions sample a duration value from their duration distribution and the one with the lowest duration fires next.

Note that additionally to these global policies that apply for the entire model, the discussion by Marsan also includes various forms of mixed policies that allow specifying the execution policy per marking. Mixed policies remain out of scope of this thesis, as it adds more complexity on the model level. When using the race policy in our model, we can still add decisions that are based on transition weights, if required—by modelling these decisions explicitly with immediate transitions.

Also note that even if the race policy is used in GDT_SPN models, preselection applies to immediate transitions that are enabled at the same time. In the case that transitions race for the right to fire, multiple transitions make "progress" concurrently. There has to be made a choice regarding the memory of timed transitions that lose the race. Different memory policies exist:

AGE MEMORY In this memory policy, transitions keep and accumulate their progress until they fire. That is, they sample a duration once they get enabled and continuously subtract passed time (as long as they are enabled) until their time reaches zero and it becomes their turn to fire. Intuitively this can be understood as starting work once enabled, and pausing it, when disabled. No progress is lost in this memory policy.

ENABLING MEMORY In this memory policy, transitions keep and accumulate their progress until they fire or are disabled by another conflicting transition firing. This policy is useful to capture situations between exclusive transitions, where progress in one transition is lost, once the other fires.

RESAMPLING In this memory policy, all progress of transitions is lost whenever any transition fires. Though theoretically interesting, this memory strategy has little practical relevance for business process modeling, as all transitions in parallel branches of the process also lose their progress and restart their work, whenever one of the parallel transitions fires.

For each of these execution policies, the algorithm to enrich Petri net models with stochastic information needs to be adopted. We cover the technical details later in Sect. 3.4.

*Ambiguous Alignments*

Sometimes, if the trace does not exactly fit the model, there are multiple ways of aligning the trace to the model. Suppose a trace encountered in the log is $\langle A(0.0), B(1.6), C(1.8), D(1.9)\rangle$ and we want to align this trace to our running example model in Figure 7 (Figure 17 on page 45 shows the same model with

additional stochastic annotations). Observe that the trace contains both events for transitions $\mathbf{t}_C$, and $\mathbf{t}_D$, although they are exclusive according to the model.

We do not know exactly, which event in the log is a mistake. Perhaps even both events took place in reality. However, we assume that the model captures the as-is process. Therefore, we need to make a choice how to map the trace to the model. The cost-based alignment without any additional specification—all model moves and log moves sharing the same positive cost, synchronous costs set to 0—yields the following two optimal alignments:

| *log* | A(0.0) | $\gg$ | B(1.6) | C(1.8) | D(1.9) | $\gg$ | $\gg$ |
|---|---|---|---|---|---|---|---|
| *model* | A | $\tau$ | B | $\gg$ | D | $\tau$ | $\tau$ |
| | $\mathbf{t}_A$ | $\mathbf{t}_1$ | $\mathbf{t}_B$ | | $\mathbf{t}_D$ | $\mathbf{t}_2$ | $\mathbf{t}_3$ |

| *log* | A(0.0) | $\gg$ | B(1.6) | C(1.8) | D(1.9) | $\gg$ | $\gg$ |
|---|---|---|---|---|---|---|---|
| *model* | A | $\tau$ | B | C | $\gg$ | $\tau$ | $\tau$ |
| | $\mathbf{t}_A$ | $\mathbf{t}_1$ | $\mathbf{t}_B$ | $\mathbf{t}_C$ | | $\mathbf{t}_2$ | $\mathbf{t}_3$ |

Both these alignments have the cost of a single log move. In the first alignment, a log move of event C, and in the second alignment, a log move of event D. Note that by default (i.e., all log moves and model moves sharing the same cost) the alignment algorithm would always prefer one alignment over the other. It is not determined, which alignment will be the output, if there exist multiple alignments with equal cost. Suppose that in the process, the event D (the transition $\mathbf{t}_D$ in the model) is only happening in one out of five cases, and event C happens in the remaining four cases. If the alignment algorithm returns always the second alignment for some internal reason (i.e., it discards event D in favor of C), we will get the correct alignment only in twenty percent of the cases this trace is observed.

Because we need to make a choice between alignments with the same cost, the favorable choice is to pick the alignment which is more likely. Therefore, we count the occurrences of the events in the entire log, and order the costs of the model moves and log moves by adding small fractional costs, such that the cost of a log move of a more frequent event is cheaper than the respective cost of a less frequent event. Thereby, the cost-based alignment always returns the more probable alignment.

Obviously, this solution is a heuristic, and we cannot prevent wrong results. But by favoring more frequent events in alignments, we make the right choice more often than by letting the alignment algorithm decide arbitrarily. We have no formal proof for bounds on the probability that the choice of events is correct for this heuristic. However, intuitively, in case there are only two options to pick from, we can say that the probability of picking the correct one is at least 50 percent. Latter describes the worst case scenario, i.e., to have to pick between two equally frequent events. If one event is more frequent than the other, the probability to make the correct choice also increases.

Concluding, we remark that these conflicts only occur, if the trace does not exactly fit the model. Therefore, if we make sure that the model captures the observed behavior before annotating it with stochastic information, we can largely

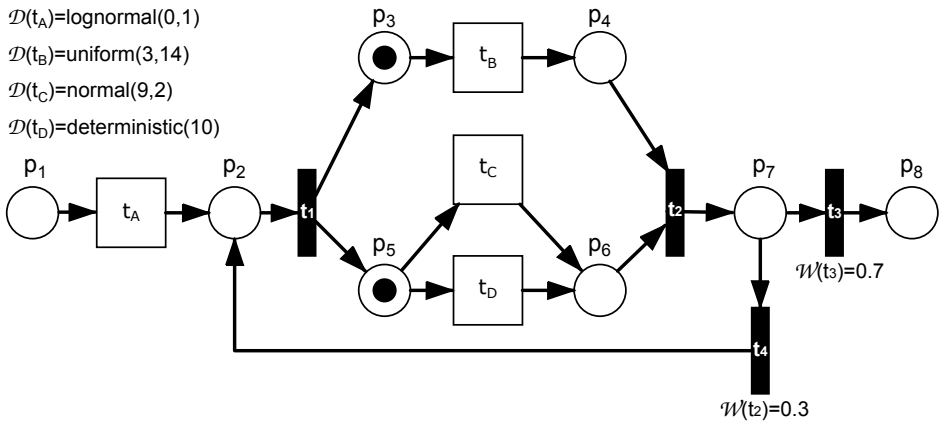



Figure 17: Example model with stochastic annotations. The duration distributions are depicted in the legend in the top left of the figure, e.g., transition $\mathbf{t}_B$ is uniformly distributed in the range $[3, 14[$. The immediate transitions $\mathbf{t}_3$ and $\mathbf{t}_4$ model a probabilistic choice, i.e., the probability to leave the loop is 0.7, and the probability to stay in the loop is 0.3.

avoid this particular ambiguity between different choices of alignments. Note that the solution we propose here does not consider the *time* of the events. Latter would provide an even better heuristic, but would require multiple iterations of the algorithm until convergence. We shall return to this idea in the outlook in Sect. 3.6.

*Censored Values*

We can extract the time that a transition takes for firing in a trace, by computing the difference between the moment of firing and the moment of enabling of that transition. If we do this repeatedly for a lot of occurrences of the transition firing, we get an overview of the distribution of that transition, e.g., we can compute the mean duration, the variance, or plot a histogram.

Depending on the chosen execution policy, however, it might happen that a transition loses a race, and loses its progress during replay of the trace in the model. In that case, we are unable to collect the actual duration of that transition, because the transition in the model restarts the work. In the following, we explain this subtle but important issue with an example.

Figure 17 shows our running example model with annotated timing information. Note that time units are omitted from the model, as they are not relevant to the discussion. We assume time units to be days unless otherwise specified in this thesis. Also note that in the current marking the three transitions $\{\mathbf{t}_B, \mathbf{t}_C, \mathbf{t}_D\}$ are enabled. Transition $\mathbf{t}_B$ is parallel to transitions $\mathbf{t}_C$ and $\mathbf{t}_D$, while latter are in conflict for the token on place $p_5$. Assume that we do not use a preselection policy, but the decision, if either $\mathbf{t}_C$ fires or $\mathbf{t}_D$ fires is determined by race. That is, whichever takes less time, will be the one to fire. Following this policy, we take a random sample for the duration of each enabled transition, and compare their values.

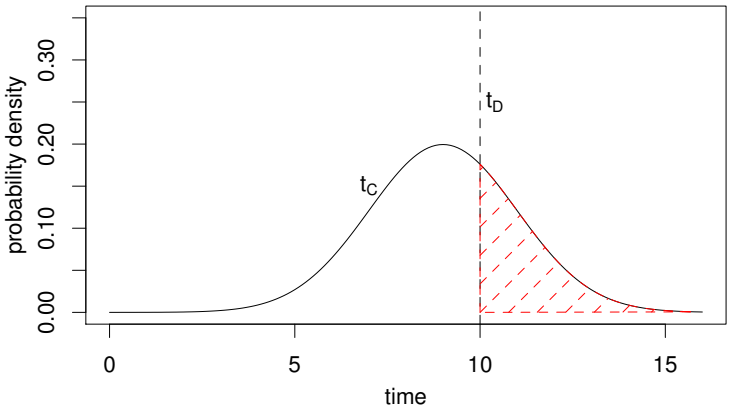



Figure 18: Density distributions with time-out. The two densities of the probability distributions of transitions $\mathbf{t}_C$ and $\mathbf{t}_D$ are depicted. Because in a race selection policy, the next transition is the one with the lower duration, transition $\mathbf{t}_C$ will never exhibit values higher than 10 (the deterministic time-out modeled by transition $\mathbf{t}_D$).

In our example model in Figure 17, the duration distribution of $\mathbf{t}_C$ (i.e., $\mathcal{D}(\mathbf{t}_C)$) is normally distributed with a mean of 9 and a standard deviation of 2. The distribution of $\mathbf{t}_D$ is deterministic and fires always after 10 time units. This model captures a time-out, as encountered in many time-constrained business processes. The time-out construct is supported by major modeling and execution languages, e.g., BPMN [150], BPEL [101]. Time-outs also have been discussed by Koehler et al. in [112].

Figure 18 shows the two probability density functions of the transitions $\mathbf{t}_C$, and $\mathbf{t}_D$. Note that while the normal density function follows the well-known bell-shaped curve, the density of the deterministic transition $\mathbf{t}_D$ is a Dirac delta function, i.e., all probability mass is concentrated on a single point (10 days). In the figure, the area right of the deterministic time-out is shaded. Be aware that whenever the transition $\mathbf{t}_C$ would take longer than transition $\mathbf{t}_D$, i.e., the random samples fall into the marked region, these values are not observed, i.e., they are *censored*.

To reconstruct the underlying stochastic model, we need to take these *censored* values into account. While in this example, the values are censored always at a deterministic boundary, more generally, the problem is to infer the underlying duration distribution, given randomly censored samples of a random variable. This problem has been investigated in the seminal work by Kaplan and Meier [104], and is also known as "density estimation with randomly censored data" [13]. Amongst others, Kooperberg provided implementations of such an estimation algorithm for the statistics software R [114].

Not taking these censored values into account would introduce a bias into the parameters of the model. Reconsider Figure 18, where all sample values of the underlying normal distribution, which are higher than 10, are censored. Here, theoretically, around a third of the samples would fall into the censored area behind the time-out of transition $\mathbf{t}_D$.

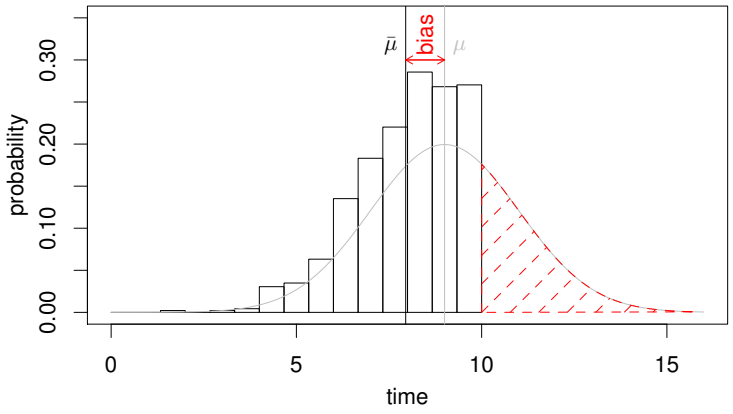



Figure 19: Bias of censored data. If we do not account for the unobservable part above the time-out threshold, the sample mean $\bar{\mu}$ is shifted to the left of the real mean value $\mu$ and produces *a bias* of about one time unit in this example.

Figure 19 shows the bias that is created, when only the observable data are used for learning the stochastic parameters of transition $\mathbf{t}_C$ in our example. The histogram shows the results produced by the remaining portion of 1 000 samples drawn from a normal distribution. In the figure, the real parameters of the distribution of $\mathbf{t}_C$ are depicted in gray as the bell-shaped curve, with the real mean value $\mu = 9$. The sample mean $\bar{\mu}$ (at around 7.97) is depicted as a black vertical line and the bias between these two quantities is shown in between. The extent of this bias depends on the competing transitions durations, and it can be difficult, or sometimes even impossible to restore the original population parameters. Think of an extreme case, where a timeout is never happening, or always firing, i.e., we do not have any concrete duration samples of a distribution, but only a lower bound.

*Sampling Bias*

When conducting a survey to get accurate estimates about a population, it is important to use a representative subset of the population. Therefore, the sample needs to be selected randomly. A comprehensible example is when participation in a survey is voluntary. Then, people with strong opinions are more likely to respond and cause a bias by underrepresenting people with indifferent opinions. Another example is to select only a certain group of cases, or perform the study in a specific region.

Translated to our setting, sampling bias occurs, when we gather performance data of a process only in certain cases. In our architecture, we do not distinguish cases but collect all. However, if an organization relies on manual process documentation, it might be that certain cases with specific characteristics (e.g., high workload in the system, or long and tiring procedures) are less likely to get documented, as the process participants are more likely to make errors under stressful conditions. We shall investigate this matter of missing documentation in Chapter 5.

Another kind of sampling bias can occur, when we collect cases of a nonrepresentative time span. Then, we select only samples of the process in the particular *state* in that time span. For example, if we elicit a stochastic model and use only the cases of the Christmas sale period, these estimates might not be representative for the entire year's process performance.

This bias can occur in case of changes in the underlying process over time. In the data mining community this is called *concept drift* (gradual change), or *concept shift* (abrupt change) [19, 208, 200, 220]. This means that the data that was used for training a model does not necessarily match the actual data for which the model is applied [85].

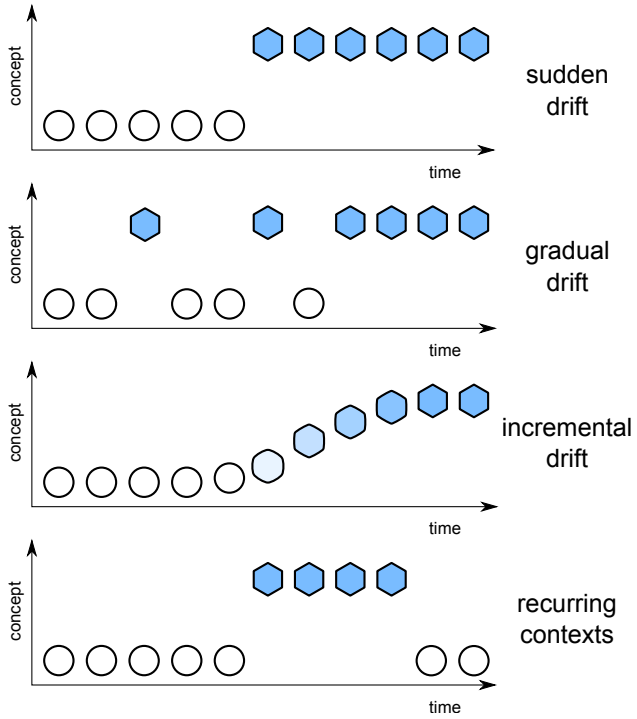



Figure 20: Concept drift types. The figure shows different classes of concept drift, cf. [220]. From top to bottom, the changes in the contexts are sudden drifts, gradual drifts, incremental drifts and recurring contexts.

Figure 20 shows different types of changes that might occur in real systems. The classification is adopted from [220]. Sudden drifts occur for instance when a migration to a new system is performed. For example, if a hospital decides to adopt a new hospital information system, the performance of the processes are expected to change abruptly. Gradual drift might occur, when the adoption of a new system of process is accompanied by a transition phase, i.e., both concepts run in parallel, but gradually less and less instances follow the old schema. Incremental drifts model smooth transitions between concepts, as might be in the case of a new process participant getting gradually accustomed to the tasks and increasing the performance until it reaches a steady state. Recurring contexts are capturing seasonality, e.g., summer and winter, or day and night.

Typical methods to adapt the model to these changes include (1) *sliding windows* [56] (only using a fixed size of the most recent observations for learning),

(2) *exponential smoothing* [72] (similar to sliding windows, but using all historical information by weighting recent observations higher than past observations), or (3) *seasonal forecasts* (autoregressive integrated moving average (ARIMA) models [42, 203]).

Sampling bias in general, and concept drift and concept shift in particular, are out of scope of this thesis, as this is a research field on its own. We assume that either the process is in a steady state, or methods to detect and adapt to changes are used in addition to the methods proposed in this work. We shall return to this limitation in Chapter 9.

*Special Cases*

We highlight special cases that we need to consider, when eliciting stochastic information of event logs and process models. First, we need to identify immediate transitions, i.e., transitions in the model that serve as routing nodes only. Second, we want to be able to identify deterministic timed transitions, e.g., transitions that capture fixed time-outs.

In technical terms, once we have gathered the observed samples and censored samples, we can check, whether a transition is *deterministic*, by comparing the observed samples. If the samples are sufficiently close, we define the transition to be deterministic. This can be made more robust against noise, by removing outliers before applying these rules. In the mined model, the deterministic value is estimated as the mean (or the more robust median) of the observed values. Additionally, we check, whether the censored values are consistent with the deterministic value. Recall that a censored value in our case is always of the form that a value is greater than an observed value of a transition that won the race. Thus, we need to check, if the censored values do not contradict the observed deterministic value.

Also deterministic, but not taking time are immediate transitions, which fire *immediately* after they have been enabled, provided that they are selected among competing immediate transitions. We assume immediate transitions to be invisible. But if corresponding events exist in the event log, we can identify immediate transitions by checking whether all their observed durations are between 0 and a small threshold. Note that we do not provide a default value for the threshold, as it depends on the domain how large these thresholds have to be. Response times of systems should be considered in the threshold, for example.

## 3.4 APPROACH AND ALGORITHM

Now, that we addressed the main challenges, we present the details of the algorithm to enrich Petri nets with stochastic information from event logs.

*Assumptions*

For the algorithm to produce optimal results, the following assumptions need to be met. These assumptions summarize the previous discussions.

EVENT LOGS Heterogeneous event sources are mapped to a homogeneous event log. This can be achieved by a monitoring architecture, see Section 2.4. That is, in this thesis, we are on the same abstraction level as process mining techniques. The events capture the termination of activities contained in the log.

EXISTING FITTING MODEL A Petri net model exists or can be discovered from the event log and fitness between model and event log is good. That is, we are able to map events to transitions in the model. The event log specifies, to which transition an event belongs.

STEADY STATE The process generating the data in the event log is in a steady state. This assumption is a strong one, but concept drift detection is a research topic that is out of scope for this thesis.

INDEPENDENCE We assume that dependencies between activity durations are not existent or negligible. We also assume independence between routing decisions through the process model that are made in a case.

COVERAGE There are sufficiently many traces in the event log to accurately estimate the decision probabilities, and parameters of distributions, for each transition.

### *Algorithm to Discover* GDT_SPN *Models*

Given these assumptions, the algorithm enriches Petri net models with stochastic execution information as described in Algorithm 1.

First, the costs of asynchronous transitions is determined in lines 2. The method `countTransitions` orders events by their number of occurrence in the event log, and assigns costs to the corresponding transitions, such that the ordering of the costs is consistent with the number of occurrence (the most encountered event is assigned the least cost). We do not prefer an alignment with two asynchronous moves over an alignment with one asynchronous move, however. Therefore, we make sure that the highest cost of a transition is less than twice the cost of the transition with the lowest cost.

Second, the structures to collect performance information (i.e., duration of activities and occurrence counts) during replay of the traces are prepared in lines 4–5 of Algorithm 1. Using the alignments, which we find with the techniques in [15], these structures are filled during replay of the model. Thereby, the execution policy of the GDT_SPN model is considered. During replay of each trace (line 8), we keep track of enabling times and firing times of transitions, and reset them depending on the memory policy. Hereby, we obey the execution policy and collect the enabled times of the transitions until firing. The effect of the REPLAY function is that we collect duration information of transitions. Further, if transition conflicts have been resolved by weights, we obtain relative counts of transition firings for each visited marking. These counts are incremented each time a transition is chosen in a marking.

**Algorithm 1** Enrichment algorithm

```
1:  procedure enrichPN(pn, log, policy, distType, thresh)
2:      trCosts ← new Map of String to Float
                                  // Collects transition costs for the alignment in trCosts
3:      trCosts ← countTransitions(log)
                  // Counts event occurrences, orders them, and adds fractional costs
4:      trTimes ← new Array of Lists
                                  // Stores transition durations in trTimes during replay
5:      trCounts ← new Array of Maps
                                  // Collects transition counts per marking in trCounts
6:      for all trace ∈ log do
7:          alignment ← align(trace, pn, trCosts)          // Get alignment, cf.[15]
8:          replay(alignment, trace, pn, trTimes, policy, distType)
                      // Collects information of executed transition counts per marking
                      // and transition durations respecting the chosen execution policy
9:      end for
10:     weights ← optimizeWeights(trCounts)
                  // Compute a map of Transition to Float by minimizing the squared
                      // error in each equation for each marking by gradient descent.
11:     spn ← new GDT_SPN
12:     spn.copyStructureOf(pn)
                                  // Fill new model with elements of the Definition 1
13:     for all tr ∈ spn.getTransitions( ) do
14:         times ← trTimes.getAll(tr)
15:         if allImmediate(times, thresh) then
16:             tr.setImmediate( )
17:         else if allDeterministic(times, thresh) then
18:             tr.setTimed(fitDist("deterministic", times))
19:         else
20:             tr.setTimed(fitDist(distType, times))
21:         end if
22:         tr.setWeight(weights.get(tr))
23:     end for
24:     return spn                                  // Enriched model with path
                                                    // probabilities and distributions
25: end procedure
```

We require the transition counts per marking (variable trCounts in Algorithm 1) to estimate the weights of the transitions. This is important for the *global preselection* firing policy, where the next transition is always determined by the relative weights of the enabled transitions (cf. Section 3.3 for the discussion on different execution policies). We explain the optimization of the weights (by the optimizeWeights function) in the next subsection. After the weights have been determined, we create a new GDT_SPN model in line 11 that the algorithm returns later. The structure of the input Petri net is copied into the new net.

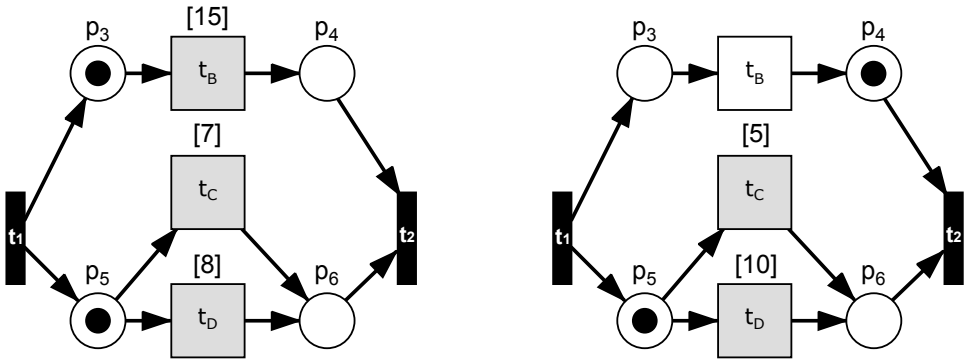
Afterwards, the transitions are annotated with their timing characteristics and their weights. For timing, we need to decide whether the transition belongs to the set of immediate transitions $T_i$, or to the set of timed transition $T_t$. If the observed durations, which we extracted during replay, are all within 0 and a user defined threshold (thresh), a transition is assumed to be immediate, cf. line 16 in Algorithm 1.

We distinguish between random transition durations and deterministic transition durations (i.e., transitions always firing after a given duration). This check is performed in line 17. If it is positive, the timed transition is assigned a deterministic distribution, i.e., one with the Dirac delta density function set to the mean (or median) of the values. If the observed times are indeed random durations, the specified distribution type (e.g., normal, uniform, or nonparametric kernel density estimation) will be fit to the data and assigned to the transition duration. Finally, the weights will be set according to the values determined by the OPTIMIZEWEIGHTS function, which we will discuss in the following. Once all transitions are updated with the stochastic information gathered from the log, the algorithm returns the obtained GDT_SPN model.

*Optimizing the Weights*

To determine the model parameters of the GDT_SPN model, we follow a maximum likelihood approach. That is, we select the model parameters, with which the observed data is best explained. In our case, we try to fit the weights of transitions (i.e., the $\mathcal{W}$ function in the GDT_SPN) to the ratio of observed occurrence counts. The problem is related to estimating the probability of a biased coin landing on heads from a number of coin flips. Latter problem is well known in statistics, and the maximum likelihood estimate coincides with the observed ratio of heads (e.g., if the coin lands on heads in twenty cases out of a hundred coin flips, the maximum likelihood estimate for the probability of the coin landing on heads is 0.2). Analogously, in our case if we encounter markings in the Petri net model during replay, where multiple transitions are enabled, we assign the weights of these transitions according to the relative ratio of occurrence. In simple examples, as in Figure 5, where we only have one decision, we do not need to optimize the weights, but we can assign the observed firing ratios of the transitions to the transition weights.

In the more general case, where single transitions can be enabled in multiple markings, and also be in conflict with different sets of other transitions, we need to *optimize* the weights. We illustrate this with our running example Petri net model from Figure 7. We assume that transition weights are used to determine the next firing transition (i.e., we assume that the preselection firing semantic is used instead of the race semantic). Figure 21 shows a fragment of the model and two markings. We annotated the model with the observed number of transition firings in each marking that we counted during replay of a log. Enabled transitions are depicted gray. On the left-hand side, in Figure 21a, the marking $\{p_3, p_5\}$ has been visited 30 times, of which in 15 cases the transition $\mathbf{t}_B$ fired first, in 7 cases transition $\mathbf{t}_C$ fired first, and in the remaining 8 cases transition $\mathbf{t}_D$ fired first.




(a) Transition counts in marking $\{p_3, p_5\}$

(b) Transition counts in marking $\{p_4, p_5\}$

Figure 21: Transition counts per marking. The figures show the inner part of the cycle in our running example. The enabled transitions are shaded gray and annotated with the counts of occurrences that could have been produced by replaying a log on the model. Note that the ratios of these counts are estimators for the real probability of selecting these transitions.

The 15 times that transition $\mathbf{t}_B$ fired, led to marking $\{p_4, p_5\}$. This is depicted on the right-hand side, in Figure 21b. In this marking, transition $\mathbf{t}_B$ is not enabled, transition $\mathbf{t}_C$ fired 5 times, and transition $\mathbf{t}_D$ fired 10 times.

Thus, from the example transition counts in Figure 21, we derive the following equations for the weight of each enabled transition in markings with competing transitions. That is, for marking $\{p_3, p_5\}$ in Figure 21a we derive Equation 6–8, and for marking $\{p_4, p_5\}$ in Figure 21b, we derive Equation 9 and Equation 10:

$$\frac{\mathcal{W}(\mathbf{t}_B)}{\mathcal{W}(\mathbf{t}_B)+\mathcal{W}(\mathbf{t}_C)+\mathcal{W}(\mathbf{t}_D)} = \frac{15}{15+7+8} \tag{6}$$

$$\frac{\mathcal{W}(\mathbf{t}_C)}{\mathcal{W}(\mathbf{t}_B)+\mathcal{W}(\mathbf{t}_C)+\mathcal{W}(\mathbf{t}_D)} = \frac{7}{15+7+8} \tag{7}$$

$$\frac{\mathcal{W}(\mathbf{t}_D)}{\mathcal{W}(\mathbf{t}_B)+\mathcal{W}(\mathbf{t}_C)+\mathcal{W}(\mathbf{t}_D)} = \frac{8}{15+7+8} \tag{8}$$

$$\frac{\mathcal{W}(\mathbf{t}_C)}{\mathcal{W}(\mathbf{t}_C)+\mathcal{W}(\mathbf{t}_D)} = \frac{5}{5+10} \tag{9}$$

$$\frac{\mathcal{W}(\mathbf{t}_D)}{\mathcal{W}(\mathbf{t}_C)+\mathcal{W}(\mathbf{t}_D)} = \frac{10}{5+10} \tag{10}$$

Dividing Equation 7 by Equation 8 yields

$$\frac{\mathcal{W}(\mathbf{t}_C)}{\mathcal{W}(\mathbf{t}_D)} = \frac{7}{8} \tag{11}$$

and dividing Equation 9 by Equation 10 yields

$$\frac{\mathcal{W}(\mathbf{t}_C)}{\mathcal{W}(\mathbf{t}_D)} = \frac{5}{10} \tag{12}$$

Obviously, we cannot find an assignment to the weights of transitions $\mathbf{t}_C$ and $\mathbf{t}_D$ that fulfills Equation 11 and Equation 12. Instead, we apply the following method. We add error variables $\epsilon_i \in \{\epsilon_1, \dots \epsilon_n\}$ for each equation.

$$\frac{\mathcal{W}(\mathbf{t}_B)}{\mathcal{W}(\mathbf{t}_B)+\mathcal{W}(\mathbf{t}_C)+\mathcal{W}(\mathbf{t}_D)} = \frac{15}{30}+\epsilon_1 \tag{13}$$

$$\frac{\mathcal{W}(\mathbf{t}_C)}{\mathcal{W}(\mathbf{t}_B)+\mathcal{W}(\mathbf{t}_C)+\mathcal{W}(\mathbf{t}_D)} = \frac{7}{30}+\epsilon_2 \tag{14}$$

$$\frac{\mathcal{W}(\mathbf{t}_D)}{\mathcal{W}(\mathbf{t}_B)+\mathcal{W}(\mathbf{t}_C)+\mathcal{W}(\mathbf{t}_D)} = \frac{8}{30}+\epsilon_3 \tag{15}$$

$$\frac{\mathcal{W}(\mathbf{t}_C)}{\mathcal{W}(\mathbf{t}_C)+\mathcal{W}(\mathbf{t}_D)} = \frac{5}{15}+\epsilon_4 \tag{16}$$

$$\frac{\mathcal{W}(\mathbf{t}_D)}{\mathcal{W}(\mathbf{t}_C)+\mathcal{W}(\mathbf{t}_D)} = \frac{10}{15}+\epsilon_5 \tag{17}$$

With these introduced error variables, the equations are valid for any assignment of parameters to transitions weights. We want to find the assignment of weights that minimizes the errors. Hereby, we also take into account the relative number of sample observations that we have about a certain weight, e.g., information about the weight of transition $\mathbf{t}_C$ is in Equation 14 and in Equation 16 with a total of $30+15$ observations, therefore we weigh error $\epsilon_2$ with $\frac{30}{45}$ and $\epsilon_4$ with $\frac{15}{45}$. Notice that we only have one source of information for transition $\mathbf{t}_D$ (i.e., the selection ratio in marking $\{p_3, p_5\}$) and therefore the error $\epsilon_3$ is assigned a weight of $\frac{30}{30}$, or 1. In this example, we want to minimize the following expression.

$$\frac{30}{45}\cdot\epsilon_1^2+\frac{30}{45}\cdot\epsilon_2^2+\frac{30}{30}\cdot\epsilon_3^2+\frac{15}{45}\cdot\epsilon_4^2+\frac{15}{45}\cdot\epsilon_5^2 \tag{18}$$

Note that we can substitute the error variables $\epsilon_1,\dots,\epsilon_5$ by the expressions using transition weights with the formulas from Equations 13–17. Then, our task is to search for the assignment of the weights that minimize the expression 18. This is an optimization problem.

Formally, the optimization works as follows. Let $\mathbb{L}$ be the universe of event logs, let $\mathbb{G}$ be the universe of GDT_SPN models, and let $\mathbb{M}$ be the universe of markings. During replay of the log $L_{A,TD} \in \mathbb{L}$ in the model $g \in \mathbb{G}$, we replay each trace of log $L_{A,TD}$ in the model $g$ according to the path defined by the alignment for that trace. Thereby, we collect the information how often each transition is fired in each marking. This is captured by the function $count_T : \mathbb{L} \times \mathbb{M} \times T \to \mathbb{N}_0$. An example for the setting depicted in Figure 21a is $count_T(\{p_3, p_5\}, L_{A,TD}, \mathbf{t}_B)$ is 15 for the log $L_{A,TD}$, in which transition $\mathbf{t}_B$ was fired 15 times in the marking $\{p_3, p_5\}$. We also define a function $count_M : \mathbb{L} \times \mathbb{M} \to \mathbb{N}_0$ aggregating all transition counts, i.e., $count_M(L_{A,TD}, M) = \Sigma_{\mathbf{t}\in T} count_T(L_{A,TD}, M, \mathbf{t})$.

Let $enabled : \mathbb{M} \to 2^T$ be the function returning all enabled transitions in a given marking of a GDT_SPN model. These are all the transitions, of which all input places have at least a token in the marking. We use the shorthand notation $T_M$ for the enabled transitions in marking $M$. Let $\mathcal{M}_c$ be the set of markings with competing transitions, which we encountered during replay. This set contains markings, where the set of enabled transitions contains more than one transition,

i.e., $\mathcal{M}_c = \{M \in \mathbb{M} \mid 1 < |T_M|\}$. Lastly, we need to count the number of times we observed information about a transition. Therefore, we define $countTotal_T : \mathbb{L} \times T \to \mathbb{N}_0$ such that $countTotal_T(L_{A,TD}, t) = \sum_{M \in \mathcal{M}_c} count_M(L_{A,TD}, M)$.

Then, our optimization problem can be stated as finding the assignment of weights that minimizes the squared errors, as in Equation 18.

$$\arg\min_{\mathcal{W}} \left( \sum_{M \in \mathcal{M}_c} \sum_{t \in T_M} \frac{count_M(L_{A,TD}, M)}{countTotal_T(L_{A,TD}, t)} \left( \frac{\mathcal{W}(t)}{\sum_{t_e \in T_M} \mathcal{W}(t_e)} - \frac{count_T(L_{A,TD}, M, t)}{count_M(L_{A,TD}, M)} \right)^2 \right) \tag{19}$$

In fact, we are facing a convex optimization problem, similar to linear regression. Note that instead of a regression line, we fit a single multidimensional point to the observed ratios. We motivated the optimization of the weights by the example using preselection, however, these conflicting ratios between activity weights can also occur between competing immediate transitions in the race policy.

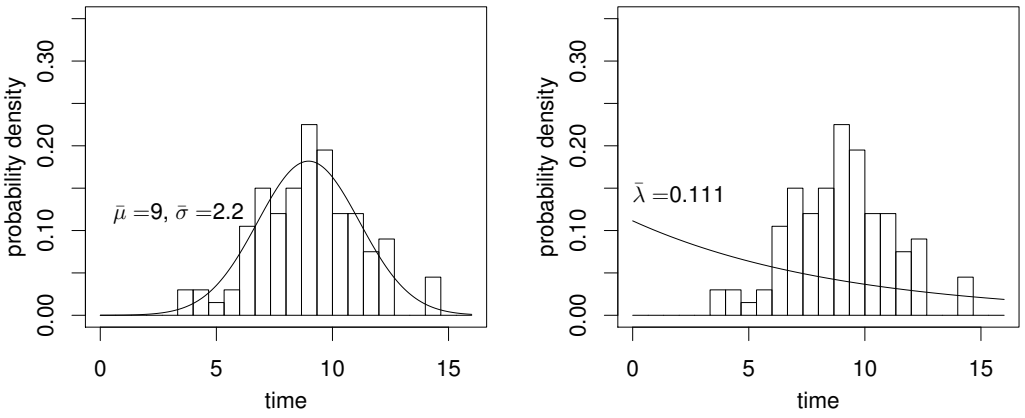
We assume that the weights of the transitions are fixed, cf. Definition 4, and therefore the error is due to chance in the observed sample. The problem that we need to solve is to find the transition weights, which explain the potentially contradicting observations best. In other words, we want to find maximum likelihood estimators for the weight parameters.

The best estimators are those that average out the errors best. In the example in Figure 21, we would try to find weights for transition $t_C$ and $t_D$ that lie between the two ratios $\frac{7}{8}$ and $\frac{1}{2}$ and minimize the error for both markings. Note that there are infinite weight pairs that specify a certain optimal ratio. For example, if $\frac{2}{3}$ would be optimal, i.e., with the weights 2 and 3 respectively, so would the ratios $\frac{4}{6}$, with weights 4 and 6, etc. Therefore, we constrain the search for optimal joint weight assignments to those weights that are normalized, i.e., the sum of the weights of all transitions in the model needs to be equal to the number of transitions in the model.

We then solve the minimization problem by a gradient descent search, see also the introduction by Snyman [194]. That is, we start with an initial assignment of weights, e.g., setting all transition weights to 1. Then, we compute the gradient vector (i.e., the partial derivation of the cost function for each transition weight). We take a step into the direction of this gradient vector with a certain step size. As we get nearer to the optimal solution, the gradient converges to 0, and we can stop the descent. Because our problem is convex, this optimization method is guaranteed to find the optimal result, when convergence is reached. In certain cases it might happen that with a large step, we move past the optimum and further away from the solution. Then, the search is diverging in each step from the optimal solution—but this can be prevented in the algorithm by controlling whether the gradient increases, and reducing the step size parameter in that case.

*Fitting Distributions*

There exist several methods to fit distributions to data. The focus of this thesis, however, is not on the invention of new fitting techniques. We already sketched the problem of the bias that occurs when trying to restore the original distributions of timed transitions in conflict. Therefore, we will only list a few examples.




(a) Normal fit to the data.

(b) Exponential fit to the data.

Figure 22: Normal and exponential fit to data. The normal distribution and the exponential distribution are fit to the same data set of 100 random samples from a normal distribution.

One of the simplest cases is to fit a normal distribution to observed data. Therefore, we only need to compute the sample mean $\bar{\mu}$ and the sample standard deviation $\bar{\sigma}$ of the data. All statistics tools support this computation, and provide built-in functions to calculate these quantities. An example is depicted in Figure 22a. The normal distribution has a property, which makes it sometimes inappropriate to model time: It is defined in the negative domain, as well as in the positive domain. For obvious reasons, we do not want to model negative times for activity durations. But the normal distribution has some benefits as well: In some models, using the normal distribution is the only possibility to derive tractable solutions. Moreover, if the data is normally distributed—as it is assumed for surgery durations [196]—this model is the best choice.

Another simple case is to fit an exponential distribution to data. The exponential distribution has only one parameter: the firing rate $\lambda$. The firing rate determines how often per time unit the transition fires. To fit an exponential distribution to data, we only need to compute the mean $\bar{\mu}$ of the data samples, and the best estimator for the rate $\lambda$ is $\bar{\mu}^{-1}$, cf. Figure 22b, where an exponential distribution is fit to a random data sample. The exponential distribution is often used to capture waiting times, or inter-arrival times of customers [44]. In this example, the data was generated by a normal distribution which makes an exponential fit inappropriate.

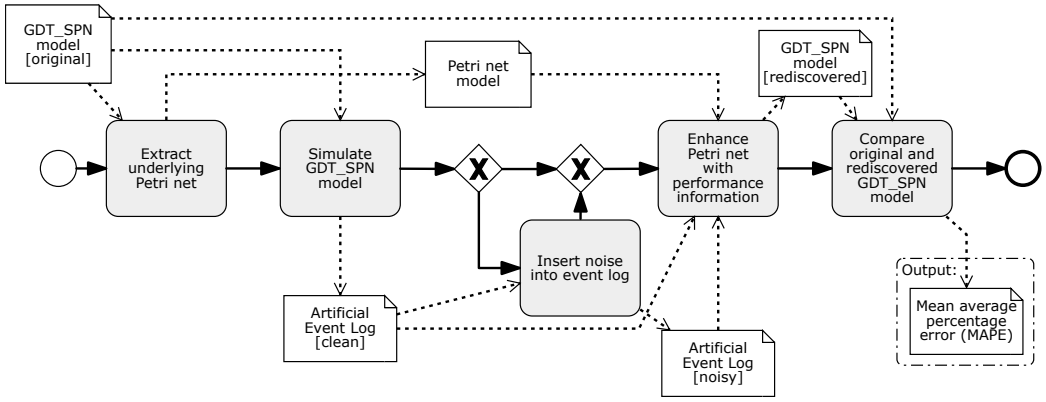



Figure 23: Setup for evaluating the quality of the discovered model. To rate the quality of the algorithm, the experiment compares the true parameters of a GDT_SPN model with the rediscovered GDT_SPN model. We analyze dependencies on available number of traces and the tolerance to noise in the event log.

## 3.5 CONCEPTUAL EVALUATION

So far, we described the challenges how we can address them, and discussed the issue of optimizing weights for transitions, and also how fitting parametric distributions to data work. In this section, we perform a conceptual evaluation, that is, we test the algorithm in controlled experiments, where our assumptions—especially the *independence* assumption—hold. Therefore, we investigate how well the algorithm can restore a given model, of which we *know* the stochastic parameters. In practice however, the theoretical model behind the observable phenomena is not known.

In this conceptual evaluation, we focus on the following two questions:

- How does the number of traces affect model accuracy?
- How tolerant is the algorithm regarding noise in the log?

### 3.5.1 *Experimental Setup*

The experimental setup is depicted as a BPMN model in Figure 23. We rely on a simulation based approach. First, we need a GDT_SPN model. Note that there exist already algorithms that can discover less expressive performance models from data [24, 115], which can serve as a starting point, or a hand-made model can be used—we our running example model depicted in Figure 17. Subsequently, multiple logs are simulated from the GDT_SPN model with increasing trace count from 10 traces to 10 000 traces. The simulated event logs, the underlying Petri net $(P, T, F, M_0)$ of the GDT_SPN model, and the given execution policy are passed as input to the discovery algorithm, see Figure 23. The algorithm produces a GDT_SPN model, which we compare with the original model.

### 3.5.2 *Model Quality Results and Interpretation*

There are several ways to assess the accuracy of a model. First, we measure the model quality results of the obtained GDT_SPN models in terms of the bias that is introduced by trying to infer the original distributions from a given number of traces. Second, we artificially introduce noise into the observable traces and evaluate how tolerant the approach is, i.e., how well it can restore the original model parameters depending on the amount of noise.

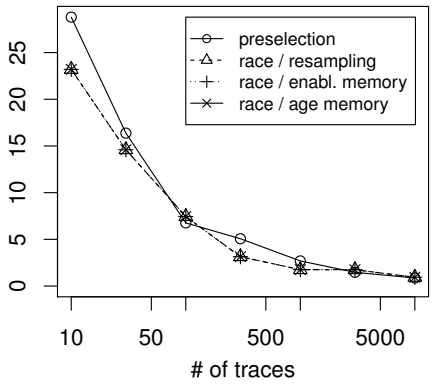
*Model Accuracy*

To test for the error that our method produces in the model parameters, we calculate the mean absolute percentage error (MAPE) of the estimated first moment and the original first moment of each timed transition's distribution, cf. the overview by Hyndman and Koehler on measures of accuracy[93]. Note that we omitted the first transition $\mathbf{t}_A$ from the calculation, because we cannot calculate its duration, as there is no previous event with a timestamp in the log. Weights are evaluated relatively to each other when selecting an immediate transition, and additionally in *preselection* mode also when selecting the next timed transition. Therefore, we need to compare the weight ratios in each marking of the original model with those of the discovered model, where selection of the next transition is based on weights. Because weights are evaluated relatively to each other, we normalize them, before we calculate the MAPE of the weights in each relevant marking.

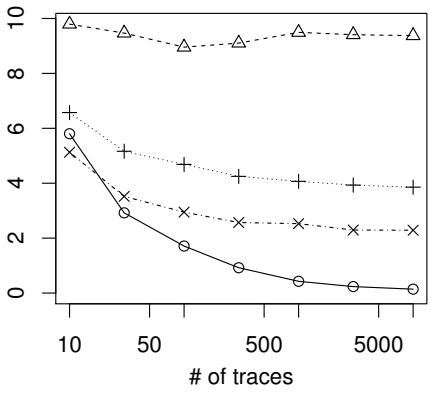
Figure 24a shows the error between the transition weights of the original model, and the transition weights of the model that we discovered from the event log. The errors gradually decrease with a higher number of traces, and the plots are in logarithmic scale on the x-axis. Note that weight errors of all *race policies* collapse, as their weights are computed in the same way. However, the *preselection* policy has more constraints on the weights, and random behavior of small event logs prohibits discovering the true weights accurately. Figure 24b shows the mean average percentage error of the 1.moments, when a nonparametric kernel density estimation is used for calculating the duration of timed transitions. As expected, the *preselection* execution policy does not suffer from bias due to censored data. The *race* with *resampling* method is the most difficult to reconstruct, as many of the samples are discarded. The *enabling memory* policy has less bias, and in the *age memory* policy, the algorithm can restore most of the original sample durations. Figure 24c depicts the error that remains, when the log-spline density estimator [114] is used. Note that this method considers censored data and can correct the bias well. It reduces the biases of the *race* execution policies significantly.

*Noise Tolerance*

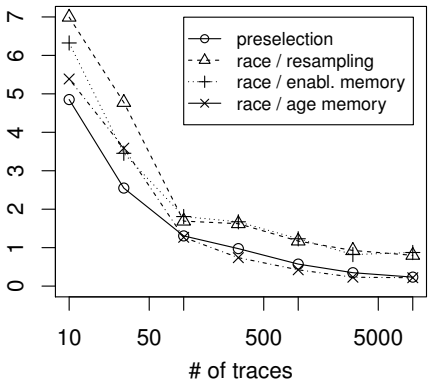
For the second experiment, we keep the trace size at 1000 and run the discovery algorithms with logs of varying degrees of artificial noise, i.e., random addition and deletion of events. Figure 25 depicts the same measures as before, i.e., the




(a) MAPE of weights.

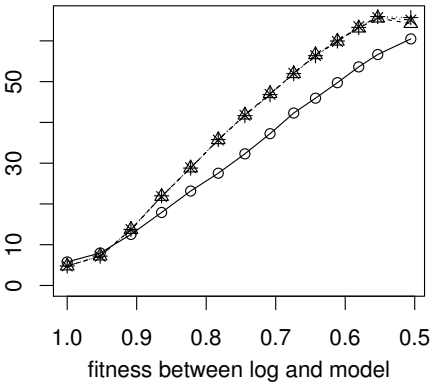



(b) MAPE of 1.moments.

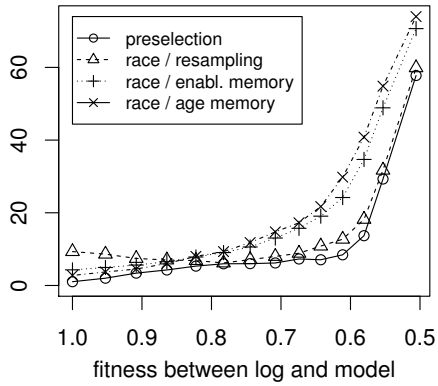



(c) MAPE of 1.moments (using censored data)

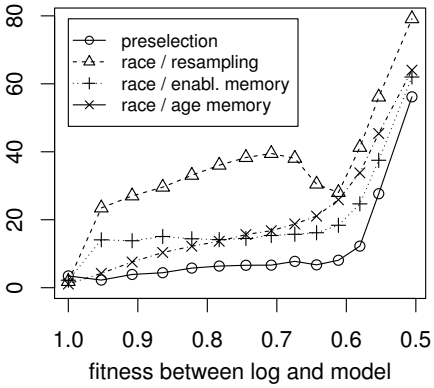
Figure 24: Effects of trace size on restored model accuracy. Mean average percentage error (MAPE) of weights and MAPE of 1.moments of inferred distributions for timed transitions of the model in Figure 17. Number of traces drawn in log-scale.




(a) MAPE of weights




(b) MAPE of 1.moments (ignoring censored data)




(c) MAPE of 1.moments (using censored data)

Figure 25: Mean average percentage errors. The graphs show the errors between the model in Figure 17 and the reconstructed model with increasing amount of noise, i.e., reduced fitness.

MAPE of relative weights in the markings and the MAPE of the 1.moments of the distributions. Observe how in Figure 25b the MAPE of the 1.moments increases non-linearly with lower fitness values. The quality starts dropping rapidly below a fitness of 0.8 in this example. When dealing with noisy logs, the Petri net models should be repaired first in a preprocessing step, as described in Section 3.2. In Figure 25c, we see that with noise, the logspline fitting method with censored data [114] fails to produce reliable results. This is caused by the failure of our algorithm to detect immediate and deterministic transitions in the presence of noise, and these special cases are treated as regular timed distributions with the specified distribution type to learn. We try to fit a log-spline density to data that consists of many equal values (many zero values for immediate transitions and many ten values for the deterministic transition $\mathbf{t}_D$) and some outliers. The log-spline fitting method has problems with distributions that are not smoothly distributed, but are mostly concentrated at a deterministic single value, and therefore the resulting values in Figure 25c with noise are not as good as expected. Additional preprocessing to align model and log in case of noise, or relying on another density estimation technique, could improve the results.

*Interpretation of the Results*

We used a small example model, cf. Figure 17, for the purposes of our evaluations. But the results show that indeed, the sample size influences the accuracy. We know from the central limit theorem [36] that the statistics that when taking random samples of a population, the sample mean $\bar{\mu}$ is a random variable—as it is based on random samples that differ every time we collect a sample—that is normally distributed with the parameters $\bar{\mu} \approx \mathcal{N}\left(\mu, \frac{\sigma^2}{n}\right)$.

This in turn states that by increasing the sample size, the variance of the mean decreases. This should be intuitive, as it becomes more likely to get the true population mean by taking more random samples. In the extreme case that we sample all members of a population, we get the true mean. As concluding remark, we caution against drawing general conclusions from these preliminary evaluations. Larger errors are expected for models with bigger state spaces.

## 3.6 DISCUSSION

In this chapter, we described the conceptual approach to enrich Petri net models with stochastic information. We proposed an algorithm and evaluated its performance by measuring the bias it produces as compared to an original model. The approach has also a number of limitations, which we want to point out here. These limitations provide hints for possible future work.

*Limitations*

Beside the discussed assumptions (e.g., steady state of the process, independence between activities), the presented approach has the following limitations.

RESOURCE AGNOSIS The current approach is unaware of resources in the process. These are only implicitly captured in the duration distributions.

INDEPENDENCE BETWEEN INSTANCES We treat each process instance individually. That is, we do not consider relationships between instances and prolonged waiting times due to other instances were started before and are being processed during execution of the current instance.

WAITING VS. EXECUTION TIME The distinction between waiting and execution time is not explicit in the definition of GDT_SPN models. Waiting time in this context is the time from enabling an activity to its beginning, and execution time is the time from beginning an activity to its termination, cf. [206, Section 3.4]. In our running example and also in our case studies, we do not have the required information to make the distinction between waiting and execution times. Nonetheless, the model allows to separately capture these two times by additional transitions, that is, an activity is captured by two transitions: one for the beginning and one for termination of an activity. This can be extended to even more fine-grained activity lifecycles, as discussed by Weske [206], while the conceptual method remains the same.

MISSING FIRST ACTIVITY DURATION The duration of the first activity cannot be recovered by our current approach, if we only have termination events of activities, because we do not know when the process started. This in turn means that the algorithm interprets the first activity as an immediate transition in the resulting GDT_SPN model. Under certain assumptions (i.e., resources are captured in the event log, and resources start the next instance immediately after they completed the previous one), this limitation can be lifted. Therefore, Wombacher and Iacob propose a method to infer the distribution of start times based on completion times of previous activities of the starting resource [212]. Their method works best in environments, where users start the next case of a process immediately after finishing the previous case.

With these limitations in mind, we want to abstract from the concrete method to a more high level viewpoint. Next, we discuss degrees of freedom of stochastic models, and smoothing techniques.

*Degrees of Freedom in the Model*

The optimization step to find the most likely weights is most helpful if the log size is relatively small, or if for certain markings there is little information available about which transitions fired. Due to the law of large numbers, the error terms will get close to zero, if there are enough observations, and the model is truly using the same set of weights for the activities in each marking. Another option is to add more flexibility to the model, i.e., to allow transition weights to depend on the markings. Latter option is providing more flexibility, but increases modeling effort tremendously.

Even more intricate is the notion of *history aware* stochastic Petri nets [191]. This model allows transition probabilities that not only depend on the current marking, but on the entire history of the current execution. This formalism helps capturing the effect in loops, where after each iteration, the probability to leave the loop rises. However, training such nets from data might become difficult, as depending on the model, the possible history of a state (i.e., the different combinations to get to a certain state) can suffer from combinatorial explosion. Consider a car manufacturing process where a customer can make $n$ binary decisions whether to pick extras like climate control, radio, electronic seat adjustments, and others. There are $2^n$ possible histories in such a case. To train such models can become difficult, as the number of required samples to estimate all the parameters correctly, also increases exponentially with the number of decisions. Too much expressiveness in the model can lead to overfitting issues [38].

We presented a way to learn the parameters of a simple model, where the weights of transitions are fixed and do not depend on the current—or previous—state of the execution. Technically, the presented approach is the more difficult one, as in other cases, we do not need to average weights, but can simply rely on the observed ratio of times.

*Smoothing Techniques to Avoid Extreme Values*

In this work, we proposed to use a *maximum likelihood* approach that coincides with counting the relative observed frequencies, cf. [199]. However, such a method will generally underestimate the probability of a transition which has not been observed in the event log, but is possible according to the model. In latter case, a transition weight of $0$ will be assigned to such a transition. Is it right to assign this extreme probability value, only because we did not encounter a trace in our event log that took the respective path?

In machine learning, several techniques to circumvent assigning these extreme probability values have been proposed, cf. the study by Zhai and Lafferty on smoothing methods [215]. For example, discounting methods, such as the Katz back-off model [107] or [28], reduce the counts of an observed word (in our case counts of observed transition firings) by a certain value and redistribute the "freed" probability mass to possible unobserved words.

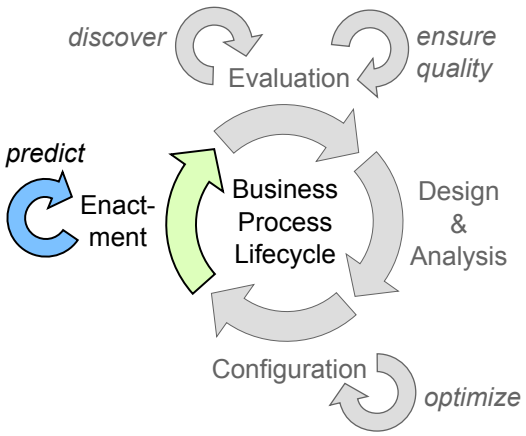
The adoption of such smoothing methods to the weight calculation appears straightforward. For example, the *add one* smoothing method—a special case of Laplace smoothing—simply increases the count of all transitions by $1$ [125, Chapter 6]. Thereby, zero values are avoided trivially. However, this simplistic smoothing method does not yield the most effective parameter estimators [49].

Let us close the discussion at this point, and turn to the first use case of the discovered performance model: The *prediction* of remaining process durations in the next chapter.

# 4

# PREDICTION OF REMAINING DURATIONS

*This chapter is based on the results published in [177].*




In the previous chapter, we described an approach to discover generally distributed transition stochastic Petri net (GDT_SPN) models. In this chapter, we turn to one of the potential applications of this type of model, that is, *prediction* of remaining duration. In the business process lifecycle, this application is useful in the *enactment* phase, where we can use such predictions for operational support for running cases. Here, we estimate an unobserved process event that is expected in the near future, that is, we estimate the time of the process end event.

## CHAPTER OUTLINE

After a brief introduction, we discuss related work in Section 4.1. The conceptual approach that we propose is described in Section 4.2. Consequently, we compare the approach against existing state-of-the-art approaches in Section 4.3. Last, we discuss the findings and potential improvements in Section 4.4.

## INTRODUCTION

To deliver products and services in time, companies need to manage their business processes efficiently. Therefore, they track the current state of the process, to timely detect undesired deviations and react accordingly. Similarly, in the context of hospitals, it is helpful for hospital administration to have good estimates of how long patients will stay and occupy resources.

The motivation is to ensure customer satisfaction by increasing the overall ratio of products and services that complete within given time thresholds. Thus, to prevent exceptionally long remaining service times, accurate prediction methods are essential.

In this setting, we propose to use GDT_SPN models to predict process remaining time, taking into account all available information in a monitoring setting, that is, the information from already observed events and the information about the absence of expected events—up to the current point in time.

## 4.1 RELATED WORK

A lot of related work exists that deals with prediction based on historical observations, or based on initial beliefs. Prediction based on time series data is widespread. Time series capture periodical measurements of data. For example, the daily closing value of a company share at the stock market is a time series, or the weekly sales numbers of a product. Many methods exist to create appropriate models to capture the underlying process that emits such data. Due to space limitations we cannot give an overview about all of them, but instead refer to an overview in [79]. Methods investigating time series often assume changes in the state of a process over time and also a strong dependence between consecutive data points, whereas in our GDT_SPN model we assume a steady state of the process and independence between activity durations. Work on the analysis of trends and change points in processes can for example be found in [214], but are out of scope of this thesis. For this work, we assume that the current performance model of the process is representing the current real world performance, cf. Chapter 3. So either the process is in a steady state, or mentioned methods, as in [214], are used to detect concept drifts and to keep the model up to date.

### *Process Related Predictions*

Methods to analyze discrete time SPN models have been investigated by van der Aalst et al. [6] and by Reijers in his PhD thesis [166]. In contrast, we consider continuous timing in our GDT_SPN models and also take the elapsed time of an individual case into account.

There has been work on prediction of case durations based on historical observations. In their work, van der Aalst et al. use the available information in logs to predict the remaining duration based on observed durations in the past [10]. They create an annotated state transition system for the logs, which can be calibrated in terms of abstraction. In that state transition system, they collect re-

maining durations for each visited state from the traces in the log. Our approach is similar in the sense that it also abstracts from data and resources, but uses GDT_SPN models instead of transition systems, making our approach more accurate when parallelism exists in the process. The work in [10] serves as one of the benchmarks for the prediction method proposed in this thesis.

Building on the work in [10], Folino et al. [70] present an improvement based on predictive clustering. They make use of additional contextual information of a trace (e.g., the current workload in the system) to perform clustering. The idea is to group similar traces and base predictions for new ones with similar features on only similar historical cases of the log. They use the predictions to warn in case of a predicted transgression of a threshold. It seems promising to combine the work in [70] with a GDT_SPN approach.

Other work for prediction of performance was presented by Hwang et al. [92] and similarly Zheng et al. [217]. They use formulae to compute quality of service criteria, such as expected durations of compositions. Typically, these works assume the service compositions to be composed of building blocks, that is, of blocks with single-entry single-exit, cf. the formal definition in the work by Kiepuszewski et al. in [109]. The methods proposed can be used for business processes, too. However, the block-structured assumption is lifted in this work, allowing for more complex models, and we also consider already running instances.

The work presented by Leitner et al. [116] also considers running instances. They use regressions for durations between two-point measures in the process. The predictions are then used to identify whether a service level agreement will be violated. By contrast, our work includes knowledge of the whole business process model to make predictions and we use the elapsed time since the last event as constraining factor.

Closely related to our approach is the prediction method presented by Wombacher and Iacob in [211]. In that work, the authors prepare event logs of unstructured processes for prediction of activity durations. They use mean duration of activities for predictions, but they do not use runtime information of elapsed time since the last event.

Also based on the assumption that activity durations are normally distributed, the work by Anklesaria et al. estimates completion time for PERT networks [25]. PERT networks are used to model projects with required activities that are in dependency relations, e.g., an activity can only begin after two previous activities have been completed. Optionality and loops are not captured in such models, however, and the remaining project time is determined by the longest path through the network. Anklesaria et al. investigate the effects of correlations between different paths to increase the accuracy of the predicted error by taking into account multiple paths.

Kang et al. [103] advocate business process monitoring in real time. Their approach is based on classifying historical traces in correct and incorrect traces by data mining techniques, such as support vector machines. Similar to our motivation, their goal is to predict and classify current instances, e.g., if they are likely

to exceed deadlines. Their approach only captures sequential processes, however, and timestamps of events are not considered in the prediction.

Simulation has also been proposed and used for operational decision making by Rozinat et al. [184]. The idea is to set up a simulation environment capturing the current situation and start a short-term simulation from this state with different simulation parameters. Their use of simulation is for operational decision support and is focused on the overall performance of business processes. In contrast, we use simulation to make a prediction for the current instance only and use the current elapsed time as additional input to the simulation which allows to improve single predictions.

Analysis of stochastic Petri nets with generally distributed firing times (i.e, GDT_SPN models in this thesis) has already been done before. Monte Carlo simulation is the preferred choice for analysis, e.g., in [31, 218]. However, previous work rather focuses on transient analysis (e.g., average throughput and waiting times of the model) instead of predicting remaining durations of single instances with conditional probability densities.

*Quality of Service Related Forecasting*

Jiang et al. [100] handle time series in business activity monitoring and focus on detecting outliers and change points. Their approach does not consider the process structure, but they only consider specific features of a process, such as customer usage profiles, and within those features they focus on adapting the prediction model to observed trends and changes. Another approach by Zeng et al. [214] applies the ARIMA forecasting method to predict performance criteria for event sequences (corresponding to *traces* in our terminology) and thus support seasonality of changes. They do not use that approach for single instances, but rather for aggregated key performance indicators. Their prediction can be mapped to the prediction approach in [10] with the states distinguished as lists of ordered events. Based on that model the values are classified by regression to separate them into the ones that meet the thresholds defined for the key performance indicators, and those that violate them. There is a drawback of using event sequences for prediction in processes with parallelism, as there is a combinatorial state space issue with the interleavings. For a single prediction only the cases that had the same interleaving order of events are used as estimators. Much useful training data are spread to other interleaving orders and do not influence the prediction for the current sequence. Compared to that, our work improves the aspect of prediction of a single case in *real time* and is less likely to suffer from sparse training data issues in processes with parallelism.

Trend aware forecasting methods are out of scope of this thesis, and we assume the process to be in steady state. If the process performance is subject to seasonality or trends, however, an integration of trend-aware methods like ARIMA seems promising. In the following, we explain our approach in more detail.

## 4.2 APPROACH AND ALGORITHM

In this section, we introduce the prediction algorithm. The prediction is done using historical information (i.e., information on how similar cases have performed in the past) and the information that we have about the current case (i.e., the previous finished activities, and the current time).

### 4.2.1 *Assumptions*

The following assumptions are necessary, so that the prediction algorithm can produce optimal results.

AVAILABILITY Historical and current process information (i.e., the start or end of activities) are available as event logs, cf. Definition 5.

TIMELINESS The events that happen in reality and indicate process progress are timely detected (i.e., the time from occurrence of an event to the corresponding update in the event log is negligible).

SOUNDNESS The GDT_SPN model that we use for prediction is a *sound* WF-net, cf. Section 2.2. This assumption is reasonable, as usually models exhibiting deadlocks, or livelocks are assumed to be subject to modeling errors [59].

INDEPENDENCE Activity durations do not depend on each other. That is, they are mutually independent. This assumption specifies a simplified worldview, as sometimes correlations might occur between activities. For instance in a hospital surgery process, all activities may take longer than normally, if there are complications with a patient. On the other hand, there may also be negative correlations. For example, process participants might hurry, after having spent too much time on a previous task, to meet a given deadline. We shall investigate the validity of this assumption with three case studies in Chapter 8.

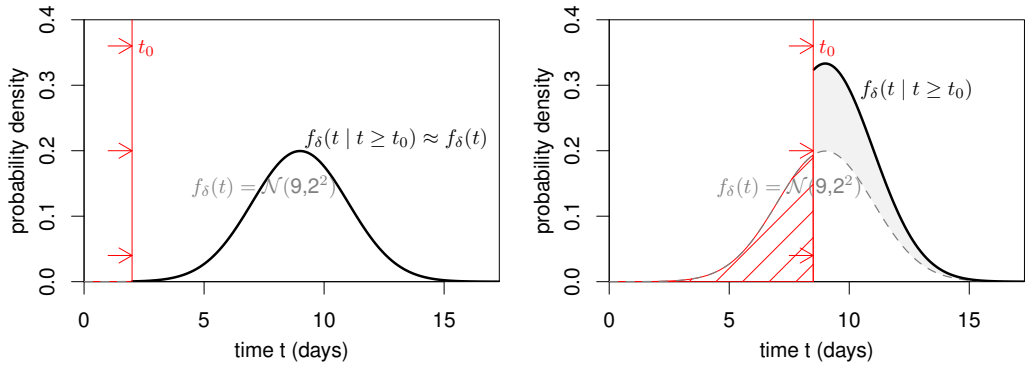
We rely on either a workflow engine to provide the *availability* of events in an event log, or in case of manual execution, we rely on a monitoring architecture, as sketched for instance in [88], cf. Section 2.4.

Now, we turn to the key concept we base the prediction upon—that is, constrained activity durations.

### 4.2.2 *Conditional Activity Durations*

Based on the assumption of *timeliness*, we can use information that goes beyond the plain list of observed events. We can use the information that a single activity has *not been completed yet* at the current time, if the event for completion of the task is not contained in the event log.

The concept is best illustrated with an example. Recall the GDT_SPN model in Figure 17 on page 45. For sake of the following discussion, assume that the execution policy of the network is *preselection* (i.e., the decision which transition




(a) After two days, the truncated density is almost unchanged

(b) After 8.5 days, the truncated density is significantly different from the original density

Figure 26: Probability density function conditioned on time. The graphs show the density functions of the duration of transition $\mathbf{t}_C$, i.e., constrained to (a) being greater than one day, and (b) being greater than 4.5 days.

is firing next in a marking is made randomly based on transition weights). We are not restricting the memory policy, as we only consider a single transition firing in this example.

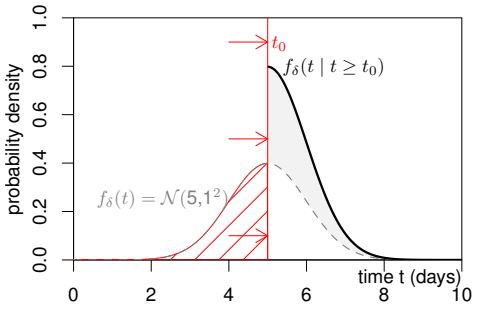
Without loss of generality, we assume that transition $\mathbf{t}_C$ was selected in the lower branch of the model in Figure 17, and we align the time axis of the probability density function of transition $\mathbf{t}_C$ to zero when it becomes enabled. Figure 26 shows the density function of the conditional distribution of transition $\mathbf{t}_C$ at two *later* points in time. Figure 26a depicts the normally distributed probability density, which is the well-known bell-shaped curve, where after two days, transition $\mathbf{t}_C$ did not fire yet. Note that this observation does *not* change the original distribution much, as it is very unlikely that the activity is completed earlier than after two days. Figure 26b shows the same situation, but more time has passed without detecting the firing of transition $\mathbf{t}_C$. In dashed gray the probability density function of the original duration $f_\delta(t)$ is depicted. The vertical line shows the current time $t_0$ that advances from left to right, as time proceeds. The thick black curve $f_\delta(t \mid t \geqslant t_0)$ is the truncated density function that represents the distribution of the activities that took longer than $t_0$. It depicts the conditional probability density of the duration conditioned on having a duration that is greater than $t_0$.

More generally, let $t \in TD$ represent time. Let $\mathbf{t} \in T_t$ be a timed transition with the assigned duration distribution function $F_\delta = \mathcal{D}(\mathbf{t})$. We obtain the *density* function $f_\delta$ by differentiating the distribution function $F_\delta$, that is, $f_\delta(t) = dF_\delta(t)/dt$. Let $t_0 \in TD, t_0 \geqslant 0$ be the current time since enabling of $\mathbf{t}_i$. Let further $f_{\delta_{Dirac}}$ denote the Dirac delta function which captures the whole probability mass at a single point. Then we define the density function of the truncated distribution as:

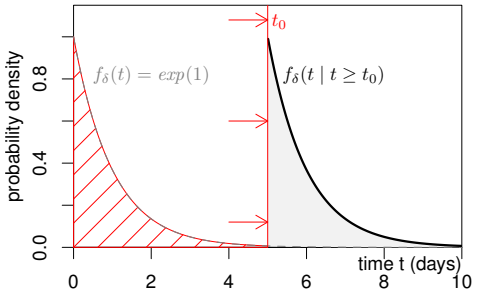
$$f_\delta(t \mid t \geqslant t_0) = \begin{cases} 0 & t < t_0 \quad F_\delta(t_0) < 1 \\ \frac{f_\delta(t)}{1 - F_\delta(t_0)} & t \geqslant t_0, \quad F_\delta(t_0) < 1 \\ f_{\delta_{Dirac}}(t - t_0) & \qquad\quad F_\delta(t_0) = 1 \end{cases} \tag{20}$$

The part of the density function that is above the threshold $t_0$ is rescaled such that it integrates to 1, which is a requirement for probability density functions. Note that in the exceptional case that $F_\delta(t_0) = 1$ (i.e., the current time $t_0$ progressed further than the probability density function's support), we use the Dirac delta function with its peak at $t_0$. In this case, the activity is expected to finish immediately at $t_0$.

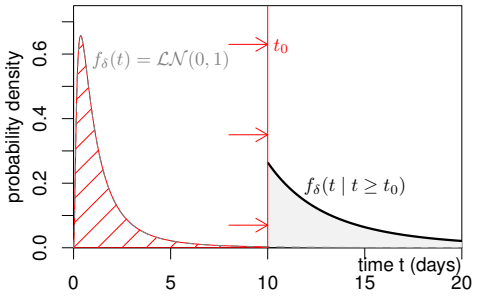
The intuition is as follows. We base our predictions on a stochastic model describing the distribution of a large amount of cases. Thereby, we discard the fraction of the cases that is not consistent with our observation for the current activity's duration, i.e., those cases that would have completed the currently running activity before the current time. In contrast to using conditional probability density functions, traditional methods predict the remaining duration of a case only upon event arrival, and subtract elapsed time from the predicted duration at later points in time [61, 10].




(a) truncated normal density




(b) truncated exponential density




(c) truncated lognormal density

Figure 27: Different truncated probability density functions.

Figure 27 shows the effect that *truncation* has on different types of distributions. The density of the normal distribution (depicted in Figure 27a) decreases faster than that of the exponential distribution. Therefore, truncation of the normal distribution makes us more certain that the corresponding event will happen soon. In the figure, the value of the conditional density function is higher than the original density and more concentrated.

The exponential distribution, depicted in Figure 27b, is a special case, where conditioning on elapsed time does not affect the shape of the distribution, that is, $exp(t) = exp(t + t_0 \mid t > t_0)$. Latter property is an effect of the memoryless property. It implies that the probability that an event will occur in the next minutes is not affected by the time that we spent waiting before. That is, if X is an exponentially distributed random variable, then for any $t, t_0 \geqslant 0$ it holds that $P(X > t_0 + t \mid t > t_0) = P(X > t)$.

Heavy-tailed distributions (e.g., the lognormal distribution depicted in Figure 27c) are on the other side of the spectrum. Latter distributions can get a

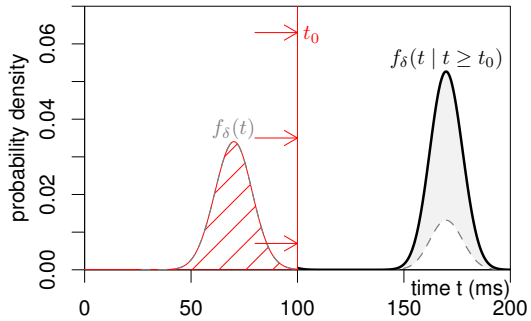



Figure 28: Conditional multi-modal distribution. When a distribution consists of two modes as depicted here (e.g., caused by two different service execution environments that are hidden to the user), the knowledge that no event occurred after waiting for 100ms makes the constrained distribution $f_\delta(t \mid t \geqslant t_0)$ more accurate than the unconstrained distribution $f_\delta(t)$.

higher variance, if we condition on elapsed time. This means that the longer we wait for an event, the less likely it will be that we will observe it in the next minutes. The uncertainty and the expected value of the distribution grows with the time that we spent waiting. In the figure the value of the conditional density is lower and decreasing at a slower rate than the unconditional one.

We can gain the most information about the expected events, when conditioning multi-modal distributions (i.e., distributions with multiple peaks in their density functions). An example is depicted in Figure 28. The depicted distribution has two modes, that is, its values are clustered at two different time points. Let us assume that the service response time of a cloud computing environment exhibits such a distribution. This could be due to fast servers (average response time: 70 milliseconds) and slow servers (average response time: 170 milliseconds) that respond to the request. Let us assume that there are three times more fast servers as there are slow ones and that a request is distributed randomly among the servers. Note that we cannot observe in advance, which type of server will provide the response, therefore, we cannot use clustering techniques for prediction in this case.

The average duration of the service is $(3 \cdot 70 + 1 \cdot 170)/4 = 95$ milliseconds in this example. If we observe that there is no response by the time $t_0 = 100$ milliseconds, we can condition $f_\delta(t)$ on this information and our estimated value of $f_\delta(t \mid t > t_0)$ becomes 170 milliseconds, which is accurately capturing the slower servers' response times. In other words, we can conclude with high certainty that the request is handled by the slower server, if we have waited for 100 milliseconds without response. Note that not taking elapsed time into account, our estimation would be based on all historical cases (i.e., captured in $f_\delta(t)$) and result in the expected duration of 95 milliseconds. This example shows that we can get more accurate results by excluding inconsistent cases from prediction, especially if we use non-parametric methods that are able to approximate distributions with multiple modes.

### 4.2.3 *Prediction Algorithm*

Besides the already mentioned assumption of immediate detection of events by the prediction framework, we consider each activity duration in isolation, independently from other activities. This is a common simplifying assumption that we share with all analytical approaches to prediction.

To make predictions for a single case needs, we need to know the current state of the case and the model that captures experiences about the behavior of the process. The prediction algorithm takes four inputs: (1) the GDT_SPN *model* of the business process, cf. Definition 4 on page 16, (2) the current *trace* of the case, i.e., all observed events up to time $t_0$, (3) the *current time* $t_0$, and (4) the number of simulation *iterations* indicating the precision of the prediction. Algorithm 2 describes the procedure.

**Algorithm 2** Prediction algorithm

```
1: procedure PREDICT(model, trace, currentTime, iterations)
2:     currentMarking ← REPLAY(trace, model)
                                   // replay the observed events in the model
3:     times ← new List()                          // used to collect results
4:     for all i ∈ iterations do
5:         time ← SIMULATECONDITIONALLY(model, currentMarking, currentTime)
6:         times.ADD(time)
7:     end for
8:     return GETMEAN(times)              // the average of the simulated values
9: end procedure
```

The algorithm is straightforward. It starts in line 2 with finding the appropriate current state (i.e., the current marking) in the model by replaying the available observed events of the case in the model. In the REPLAY method, we use the *alignment* technique (see Section 2.3) to find the state in the model that reflects the observed trace with least deviations.

In lines 3–7 the algorithm collects simulation results (i.e., completion times) of a given number of simulation iterations in a list. Each simulation run represents a sample from the possible continuations of the process according to the model. The SIMULATECONDITIONALLY method simulates continuations of the trace for the GDT_SPN model, but instead of sampling from the original transition distributions $F_\delta(t)$, it samples from the truncated distributions conditioned on the current time $F_\delta(t \mid t \geqslant t_0)$, as described in Sect. 4.2.2 before. The completion times of all simulated continuations of the case are collected and the algorithm returns the mean of these sample values. Provided enough samples, their mean value converges to the expected value of the time of the process end event.

Note that the accuracy of a prediction based on simulated samples depends on both the number of computed samples as well as the standard deviation within the samples. Therefore, we also support the mode, where not a sample size is specified, but instead the user can set required accuracy thresholds. For example, the user can let the simulation continue taking samples, until the 99 percent

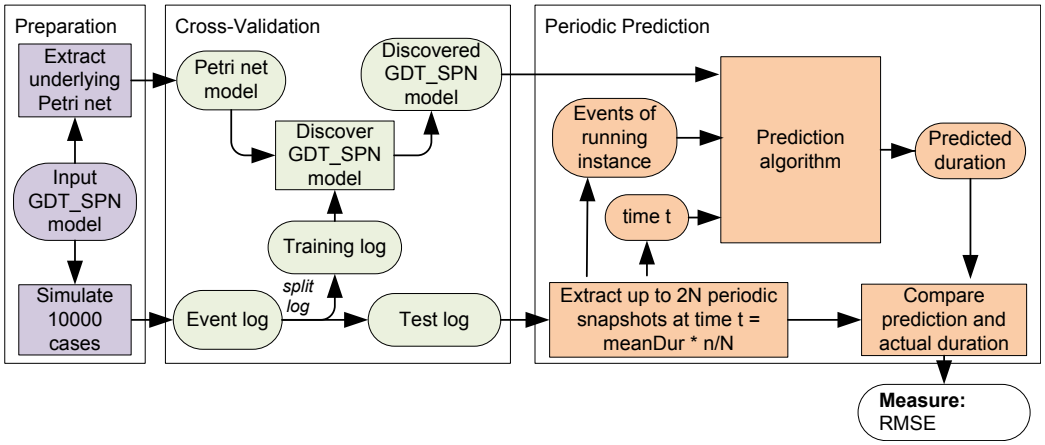



Figure 29: Setup for evaluating the prediction quality. The figure shows the *preparation*, i.e., simulation of a GDT_SPN model, and the extraction of the underlying Petri net. We use a *cross-validation*, where the log is split into training log and test log. The former is used for learning the GDT_SPN model, the latter is used for evaluating the *periodic prediction* accuracy. We measure prediction accuracy at 2N equidistant points in time, such that the mean duration is at the $N^{th}$ measurement.

confidence interval on the prediction lies within $\pm$ 3 percent of the predicted value.

## 4.3 CONCEPTUAL EVALUATION

In the conceptual evaluation, we compare our approach under laboratory conditions with other existing methods. The comparison methods are the state transition based prediction method, described in [10], and the results of a pure GSPN approach, that can be solved analytically.

### 4.3.1 *Experimental Setup*

The experimental setup is depicted in Figure 29. We want to emulate a real setting, where activity duration distributions are not known upfront, but only a Petri net model and the event log are available. Therefore, we create an GDT_SPN model with duration distributions and transition weights. From this model, we simulate 10 000 traces of execution and collect them in an event log. Besides the simulated log, also the Petri net model (i.e., the underlying Petri net of the GDT_SPN model), is used as input. Note that the stochastic information of the GDT_SPN model is discarded at this point, and the experiment only uses the event log and the Petri net model.

To evaluate the prediction quality, a 10-fold cross validation is performed. Therefore, the log is split into ten evenly divided parts and nine of them are used as the training log to learn the performance behavior and the remaining part as test log to test the prediction accuracy. We iterate over these parts, such

that each of them is used once as test log. The Petri net is enriched to a GDT_SPN by collecting the performance data in the training log.

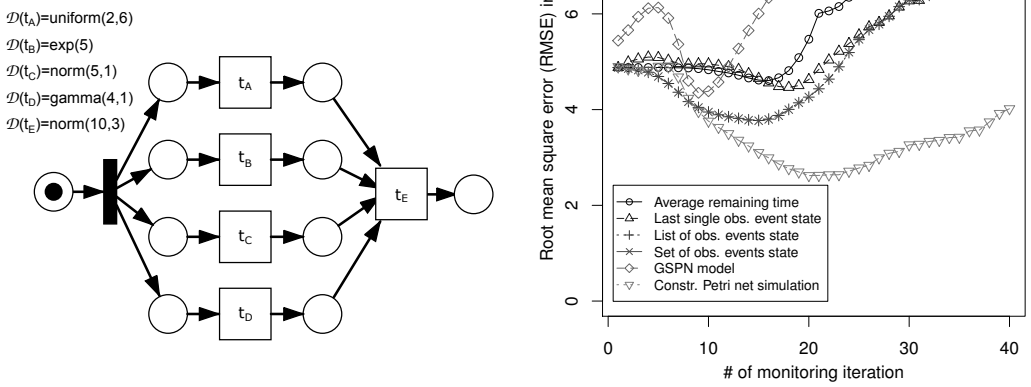
We are interested in the different prediction methods' accuracy to predict the remaining duration of a case at *any time* during the process. Therefore, we trigger the predictions periodically. The length of such a period is based on the mean process duration that is obtained from the training log. More precisely, 2N periodic predictions are made for each instance in the test log, such that the Nth snapshot is at the mean duration of the process. This means that while initially all instances still run, some instances will be finished at later prediction iterations. Note that only predictions for running cases are added to the resulting statistics.

The evaluation proceeds for each trace in the test log as follows. At each iteration of the periodic prediction, we compute the relative time $t_0$ to the trace start and pass $t_0$ and the partial trace containing the events of the case with time $t \leqslant t_0$ to the prediction algorithm described in Section 4.2.3. The predicted duration for different prediction methods are computed and compared to the actual duration of the trace from time $t_0$. The simplest method for prediction is using the *average remaining time*, which is simply the mean process duration gathered from the training set minus the elapsed time $t_0$. Additionally, we compare our predictions with the state transition systems approach [10] with different configurations. Predictions based on the history sharing (i) the *last observed event only*, (ii) the *list of all previous observed events*, and (iii) the *set of all previous observed events*. Finally, we compare our approach with a regular GSPN based approach, i.e., an approach where only exponential distributions are allowed in timed transitions of the model.

Note that if any of the methods predicts a negative remaining time, i.e., that the current case should have completed already at time $t_0$, the predicted remaining duration is set to 0.

### 4.3.2 *Prediction Results and Interpretation*

For our experiment, we set N to 20, i.e., we perform 40 periodic predictions. Figure 30 shows (a) a small model containing four parallel branches used for the simulated experiment, and (b) the root mean square error (RMSE), for each of the 40 periodic predictions mentioned above. The RMSE is an error measure for quantifying the error between a predicted and a real value, cf. [79]. The smaller the RMSE value, the better is the prediction in average. All prediction algorithms perform similarly at the start of an instance, except the GSPN model which fits exponential distributions to all timed transitions. Although the prediction error of the GSPN model is higher than that of the other methods, it can be analyzed efficiently. After some time has progressed, however, our prediction approach based on constrained Petri net simulation outperforms the other approaches significantly in this case.




(a) A GDT_SPN process model with four parallel activities A, B, C, D and a final activity E, with annotated duration distributions.

(b) Root mean square errors (RMSE) in minutes at 40 periodic predictions. Mean duration: 17.32 minutes.

Figure 30: Prediction quality for a parallel model. A model with four parallel branches (a) and corresponding prediction errors (b) using 10-fold cross-validation with 40 periodic predictions, s.t. the 20th iteration is at the mean duration.

## 4.4 DISCUSSION

Our idea for using the information of an event *not occurred yet* is tailored to processes with only sparse execution information, as motivated in Section 1.1. Thereby, we focus on situations that often occur in manual process environments, such as hospital treatment processes. Obviously, if we had a process with a thousand activities that each only take relatively small times in relation to the process duration, the conditioning of single activity durations to passed time would not yield significant improvements compared to existing prediction approaches. In summary, we collected the following insights:

MEMORYLESS ACTIVITY DURATIONS Our method cannot improve the predictions for remaining process durations, compared to existing GSPN-based approaches, if the duration distributions in the process activities are exponentially distributed. In this case both the conditional prediction method with GDT_SPN models and the GSPN-based approach produce the same result.

PARALLELISM When activities are executed in parallel, the number of allowed interleaving execution orders grows exponentially in the worst case. This means that prediction methods, that only use those historical cases, which had the observed execution order, only consider an exponentially shrinking fraction of the historical observations. Our approach that is based on the Petri net formalism does not suffer from this issue.

DENSITY OF MONITORING INFORMATION We expect less relative improvement of our prediction method in processes with a lot of monitored activities.

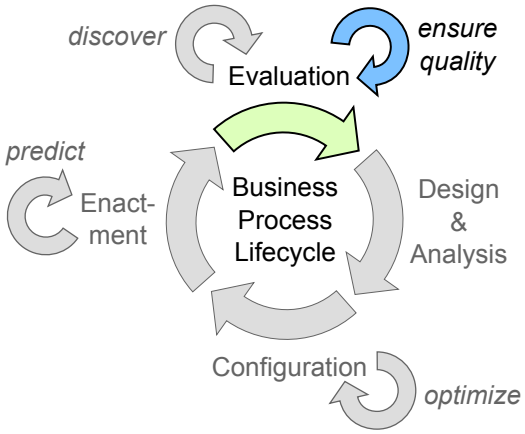
The potential improvement of our method (i.e., to improve the prediction of already running activities) is limited to the activities currently active. Therefore, an increasing number of observable activities in the process reduces the effect of individual activity durations on the process duration.

INDEPENDENCE ASSUMPTION So far, we evaluated the approach under laboratory conditions, where our assumptions—especially independence between activity durations—are satisfied by design. An evaluation with real data is required to show the potential improvements in real cases. We investigate the performance of the prediction algorithm in case studies in Chapter 8.

We conclude that, given the stated assumptions, the prediction of remaining durations can be significantly improved compared to state of the art approaches. In manual process execution environments (e.g., hospitals) monitoring information is sparse. Therefore, the presented approach can help to improve the predictions. Next, we address the problem of missing documentation that we encountered in hospitals.

# 5 IMPUTATION OF MISSING DATA

*This chapter is based on the results published in [174, 173].*




So far, we have seen how we can obtain stochastic process models of execution from data (Chapter 3) and use the models subsequently in predicting the remaining duration of a case (Chapter 4). In our assumptions the monitoring was complete—that is, we assumed events to be always detected. Unfortunately, in real settings, the event logs are often subject to quality issues [4] that affect process monitoring or process mining techniques.

In this chapter we loosen this assumption and discuss a problem that often occurs in manual process execution: Missing documentation. This problem belongs to the *evaluation* phase of the business process lifecycle, because the quality of the documentation is evaluated and compared to the process model to find deviations in order to improve documentation quality.

### CHAPTER OUTLINE

After a brief motivation, we discuss related approaches in Section 5.1. Then, we consider the problem that we need to solve in Section 5.2. Having outlined the problem complexity, we propose its decomposition in Section 5.3. We compare the results of two different artificial models in Section 5.4, where we also discuss implications and limitations of the proposed solution.

## INTRODUCTION

Suppose you need to manage a hospital, and your staff (e.g., doctors, nurses) manually record the performed treatment steps. There exist regulations that require treatment steps to be documented [27]. Furthermore, the documentation serves multiple purposes: accounting, auditing, quality securing, clinical evidence, etc. As a hospital manager, you might be interested in accounting, as only documented treatments are reimbursed by health insurance companies.

The problem is, however, that people happen to *forget* things. In our experience with the documentation of a surgery process [110], we observed that around ten percent of treatment steps were missing from the documentation. Unfortunately, we are not able to extrapolate our experience with that limited sample to generic insights. Instead, to reason about error rates of humans, we draw on large-scale studies that other researchers have conducted.

One of the largest empirical studies on errors in medical practice is the Harvard Medical Practice Study of 1984 [43]. The researchers investigated the outcomes of more than thirty thousand patient treatments in hospitals in the state of New York. They found that in 3.7 percent adverse events occurred, of which more than every fourth is due to negligence. Regardless, of the actual error percentage, we can note that errors (e.g., forgetting to document an activity) *do* occur when humans perform activities in a process.

A more qualitative approach is pursued by Reason [164]. He compares different theories of errors—personal model, legal model, and system model—and shows that different remedies are applied in practice, depending on the theory of error [164]. In this thesis, we assume the system model, which states that errors are commonplace, and not the individual is to blame, but the system should be improved to avoid errors.

In our case, we are facing the problem that documentation can be forgotten. We do not want to blame the doctors or the nurses, but provide help to identify documentation errors. Our vision is to provide a system that detects potential errors in documentation *during* enactment and notifies the process participants of these problems. This way, the participants get reminded to amend the documentation, as long as they remember whether they have done the corresponding activity, and when exactly.

The problem of forgotten documentation is a special case of the problem of missing data. We provide a brief overview on the existing literature on this problem in the next section.

## 5.1 RELATED WORK

The term *missing data* is used, when for some cases and for some variables data is missing. The problem of missing data has been studied extensively in statistics over more than fifty years [209, 186, 21, 189]. Statisticians collect surveys from samples of a population to make inferences about parameters of the whole population. Often, values are missing from these surveys due to a number of reasons. Consider a survey asking people for their income. In this survey people

might not respond, if they do not understand the question due to language reasons, or they might not want to respond if their income is very high, or very low. Thus, when dealing with missing data, it is important to know the mechanisms causing such phenomena.

*Mechanisms Causing Missing Data*

In statistics, the *missingness mechanism* describes the manner in which data are missing from a sample of a population. The early work by Rubin [186] discovered important implications of the process leading to missing data, and analyzed which minimal conditions are necessary to ignore the missing data mechanism. He distinguishes three kinds of the mechanism causing missing data: missing completely at random (MCAR), missing at random (MAR), and not missing at random (NMAR) [186], see also the overview by Schafer and Graham on the techniques to deal with these cases in [189].

Rubin found that the data need to be *missing at random* (MAR) (i.e., the condition whether a data is missing, is not influenced by its value) for unbiased imputation of the missing data. Further, he defined the mechanism that causes data to be amiss as a probabilistic phenomenon. To clarify what MAR means, we introduce the following notation. Let $X_{comp}$ be the complete data that can be partitioned in observed and missing data $X_{comp} = (X_{obs}, X_{miss})$. Let $R$ be the random variable of missingness. This means that $R$ indicates, whether a data set contains missing entries. We can think of $R$ as a random process that has a certain probability of occurring and causing data to be amiss. The MAR property specifies that the probability of missingness is independent from the missing data:

$$P(R \mid X_{comp}) = P(R \mid X_{obs}) \tag{21}$$

This allows the missing data values to depend on the observed data, but not on the missing data. A more rigorous assumption is that the missing data mechanism is independent also from the observed values. In this case the data is *missing completely at random* (MCAR), and the following equation must hold:

$$P(R \mid X_{comp}) = P(R) \tag{22}$$

MCAR implies Equation 21 and therefore techniques applicable for the MAR case are also applicable for MCAR case. If the missing data mechanism R depends on the *missing* data, the data is NMAR. This is the most challenging case, as the missing data mechanism has to be modeled and taken into account to get unbiased estimates for the population parameters. We do not address the latter case, but refer to the overview by Schafer and Graham in [189] that discusses possible applications.

*How to Deal with Missing Data?*

Two common methods to deal with missing data are *listwise deletion* (i.e., to discard an observation containing missing values from the statistics), or *imputation*

(i.e., to insert artificial values for the missing data). The advantage of using listwise deletion is that it is simple. If the data is MCAR, and only few entries in the data have to be discarded due to missing events, this solution can be used without losing too much information. Listwise deletion has a drawback, however. If a large portion of the samples have to be discarded, the efficiency of the estimate suffers from this technique. Furthermore, if the data is not MCAR, but MAR, this method fails to take dependencies into account and produces biased parameter estimates.

Better results can be achieved with imputation methods, but special care is required. A basic imputation method is *mean substitution*, i.e., replacing missing values of a variable with the sample mean of the observed values. Mean substitution generates a bias, because the uncertainty of the missing variables is not taken into account.

More sophisticated imputation methods that are able to deal with data that is MAR, and are recommended as state of the art [189], are *maximum likelihood estimation*, and Bayesian *multiple imputation*. Both these methods are efficient (i.e., they make use of all observed data to increase confidence in the estimations), and produce unbiased estimators in the MAR case. The seminal work by Dempster et al. [58] describes the *expectation maximization* algorithm that finds the maximum likelihood of the parameters by iterating an estimation and maximization step until convergence. Later, Rubin introduced the idea to replace missing values with multiple simulated values (imputations) and average results of each data set to account for the variability of the missing data [187]. The technical details of these techniques are not in the focus of this thesis, however, and the interested reader is referred to the book by Little and Rubin [119] on this topic. The aforementioned imputation techniques, however, focus on missing values in surveys and are not directly applicable to event logs, as they do not consider control flow relations in process models and usually assume a fixed number of observed variables. In business processes with cycles, there is an additional complexity, because we need to determine how often cycles have been traversed in a case.

### *Missing and Noisy Data in Event Logs*

Related work on missing data in event logs is scarce. Nevertheless, in a recent technical report, Bertoli et al. [34] propose a technique to reconstruct missing events in event logs. The authors tackle the problem by mapping control flow constraints in BPMN models to logical formulae and use a SAT-solver to find candidates for missing events. In contrast, our GDT_SPN model builds on Petri nets, which allows us to deal with cycles and probabilities of different paths.

Methods developed in the area of process mining provide functionality that enables analysis of noisy or missing event data. In process mining, the quality of the event logs is affecting for the usefulness of the analysis results and low quality poses a significant challenge to the algorithms [4]. Therefore, discovery algorithms which can deal with noise (e.g., the fuzzy miner [83], and the heuristics miner [3]) have been developed. Their focus is on capturing the common and

frequent behavior and abstracting from any exceptional behavior. These discovery algorithms take the log as granted and abstract from rare behavior, instead of trying to identify potential missing events.

Another example is the alignment of traces in the context of conformance checking [15], which we introduced in Section 2.3. The aim of alignments is to replay the event log within a given process model in order to quantify conformance by counting skipped and inserted model activities. We build upon this technique and extend it to capture path probabilities as gathered from historical observations. Note that most of the work focuses on repairing models based on logs, instead of repairing logs based on models. Examples are the work by Fahland and van der Aalst [66] that uses alignments to repair a process model to decrease inconsistency between model and log, and the work by Buijs et al. [46], which uses a genetic algorithm to find similar models to a given original model.

In this chapter, we pursue the opposite approach, that is, we want to identify deviations from a given model and suggest to correct the supposedly faulty documentation.

## 5.2 PROBLEM AND ITS COMPLEXITY

The problem in general is to find the most likely explanations of erroneous documentation. We limit our scope to *missing* documentation, and assume that the documented events are correct. As introduced in Section 2.4, we assume that the documentation that is scattered in the IT landscape of the organization can be collected and normalized to the abstraction level of event logs, cf. Definition 5 on page 17. Thus, we can formulate the problem as follows:

PROBLEM: The problem is to find the explanations for missing data (i.e., the missing events and their time) in event logs that are most likely according to our statistical GDT_SPN model.

As opposed to a survey with a known number of variables, a GDT_SPN model can have optional branches, parallelism with many possible interleaving execution orders, and cycles. Cycles allow to repeat a set of tasks arbitrarily often, which leads to a potentially infinite number of repetitions. This means that we face a more difficult problem than identifying the most likely values of a given random variable in a survey. We simultaneously need to decide which variables are missing (most likely).

Therefore, we need to *align* a trace with the GDT_SPN model. We want to find the *most likely* alignment considering structure, priorities, path probabilities and duration distributions in the model. Note that the cost-optimal alignments in [15] that we introduced in Section 2.3, only consider the structure of the process. By incorporating into the alignment the time aspect—which is continuous in our case—the solution space of the problem, which has *countable* solutions in the structural alignment setting (countable infinite solutions, if the model contains cycles), grows to *uncountable* infinite solutions, by considering time.

In theory, we need to compare the probabilities of all possible paths in the model that are conforming to the trace. In paths that visit activities multiple

times, different assignment combinations of the events in the trace to the activities in the model exist (e.g., if there are two transitions of activity A along a path, but only one corresponding event, there are two possible assignments of the event). Additionally for each path, there are infinite possible assignments of time values to the missing events.

Let us explain the problem complexity with our running example model in Figure 17 on page 45. Consider trace $tr_3$: $\langle A(0.0), B(9.6)\rangle$ and the Petri net model in Figure 17. Clearly, the trace does not fit the model, as either the entry for activity C, or for activity D, is missing from trace $tr_3$. We assume one of the activities has happened in reality, but was forgotten in the documentation. Two cost-minimal paths through the model are given by the following (structural) alignments.

| *log* | A(0.0) | $\gg$ | B(9.6) | $\gg$ | $\gg$ | $\gg$ |
|---|---|---|---|---|---|---|
| *model* | A | $\tau$ | B | D | $\tau$ | $\tau$ |
| | $t_A$ | $t_1$ | $t_B$ | $t_D$ | $t_2$ | $t_3$ |

| *log* | A(0.0) | $\gg$ | B(9.6) | $\gg$ | $\gg$ | $\gg$ |
|---|---|---|---|---|---|---|
| *model* | A | $\tau$ | B | C | $\tau$ | $\tau$ |
| | $t_A$ | $t_1$ | $t_B$ | $t_C$ | $t_2$ | $t_3$ |

Note that by assigning equal costs to every log move, these two alignments are both cost-optimal. Furthermore, because activity B is parallel to either C or D, the missing events could also have happened before B. That is, also the following two alignments are cost-optimal.

| *log* | A(0.0) | $\gg$ | $\gg$ | B(9.6) | $\gg$ | $\gg$ |
|---|---|---|---|---|---|---|
| *model* | A | $\tau$ | D | B | $\tau$ | $\tau$ |
| | $t_A$ | $t_1$ | $t_D$ | $t_B$ | $t_2$ | $t_3$ |

| *log* | A(0.0) | $\gg$ | $\gg$ | B(9.6) | $\gg$ | $\gg$ |
|---|---|---|---|---|---|---|
| *model* | A | $\tau$ | C | B | $\tau$ | $\tau$ |
| | $t_A$ | $t_1$ | $t_C$ | $t_B$ | $t_2$ | $t_3$ |

So, by assuming that only one event is missing from the trace, as in the previous alignments, we already have four different alignments to choose from for repair. But, there might be further possibilities. It might have happened, that a whole iteration of the cycle happened in reality, but was forgotten to be documented. In this case, the path $\langle A, D, B, C, B\rangle$ would also be an option to repair trace $tr_3$. Furthermore, in the second iteration activity D could have been executed: $\langle A, D, B, B, D\rangle$, in which case two different assignments of the event B in trace $tr_3$ to the path in the model exist. In general, there are infinitely many possible traces that need to be considered for a model that contains cycles. Further, there are infinite possible time values for the inserted missing events in these traces.

To compare the probabilities of these paths, we need to compute the probability distributions of the activities on the paths and compare which model path and which assignment explains the observed events' timestamps best. To reduce

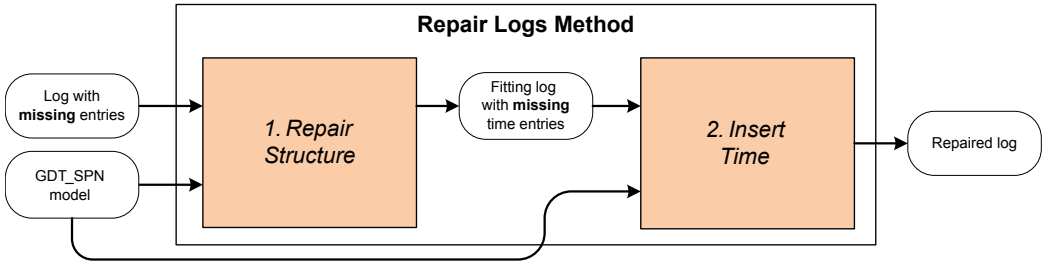



Figure 31: Dividing the problem. We divide the problem into two sub-problems: repairing the control flow and repairing the timestamps.

the complexity, we propose to decompose the problem into two separate problems: 1) repair structure and 2) insert time, as sketched in Figure 31. The method uses as input a log that should be repaired and a GDT_SPN model specifying the as-is process.

Note that by this approach, we accept the limitation that missing events on a path can only be detected, if at least one event in the trace indicates that the path was chosen. If we considered also time in the structural repair, we could identify optional events that, when added to the trace, make the other recorded events more likely. For example, if there was an optional activity that delays the next process activities by one day, we could infer whether the optional activity was executed by looking at the time that passed from the preceding activity to the next activity.

## 5.3 APPROACH AND ALGORITHM

In this section, we explain a realization of the method described above. For this realization, we make the following assumptions:

NORMATIVE MODEL The GDT_SPN model is normative. That is, it reflects the as-is process in structural, behavioral and time dimension. We assume that the process is performed according to the model.

SOUNDNESS The GDT_SPN model is *sound*, see [1] and Section 2.2.

INDEPENDENCE Activity durations are independent and have normal probability distributions, containing most of their probability mass in the positive domain. If the durations are of a different shape, we approximate them with normal distributions capturing the mean and the variance of their distribution.

CORRECTNESS OF EVENTS The events recorded in the event log happened in reality, and their timestamps are correct. All events contain a timestamp.

NON-EMPTY TRACES Each trace in the event log contains at least one event.

MISSING AT RANDOM We assume that data is MAR, that is, the probability that an event is missing from the log does not depend on the time values of the missing events.

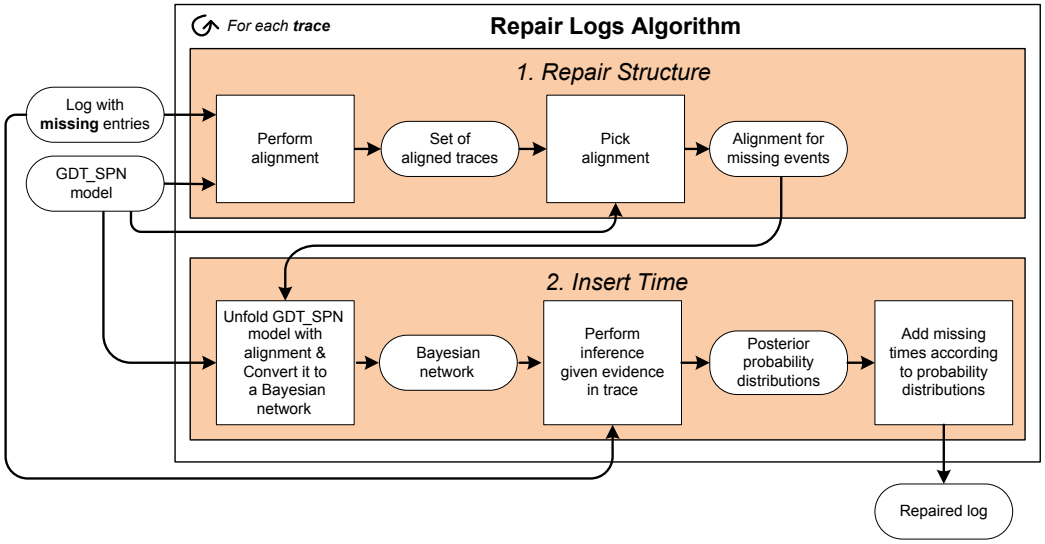



Figure 32: The repair approach described in more detail. The inputs to the algorithm are the event log with missing entries, and the GDT_SPN model. We align each trace with the model to find candidate cost-optimal alignments to choose from, based on their path probabilities. In the second step, we convert the path along the chosen alignment to a Bayesian network and infer the most likely time values to the events that are supposedly missing.

The algorithm is depicted in Figure 32, and repairs an event log as follows.

### 5.3.1 *Repair of the Structure*

For each trace, the algorithm first repairs the structure. Therefore, the path in the model is selected that best fits to the observations in the trace. The notion of cost-based alignments [15] that we introduced in Section 2.3, is used for this part. It tells us exactly:

a) when the model moves *synchronously* to the trace, i.e., where the events match the allowed transitions
b) when the *model moves* alone, i.e., an event is missing from the trace
c) when the *log moves* alone, i.e., there is an observed event that does not fit into the model at the recorded position

To calibrate the alignment algorithm so that it returns all possible paths through the model, we set the costs of *synchronous* and *model moves* to 0, and the cost of *log moves* to a high value, e.g., 1000. Then, based on the assumption of the normative model and the correctness of the events, an alignment only contains synchronous moves and model moves. This works well for acyclic models. For cyclic models, where infinite paths through a model exist, we need to assign small costs to model moves, to limit the number of resulting alignments that we compare in the next step. Notice that with this step, we consider only the alignments with a minimal number of model moves. Thereby, we exclude alignments with multiple unobserved iterations of cycles, but also exclude single optional transitions that are not reflected in the log.

In the next step, cf. box *Pick alignment* in Figure 32, we decide which of the returned cost-minimal alignments to pick for repair. For each alignment, the algorithm replays the path taken through the model and multiplies the probabilities of the decisions made along the path. This allows us to take probabilistic information into account when picking an alignment and enhances the alignment approach introduced in [15]. We also consider that, for one trace, paths with many forgotten activities are less likely than others. That is, we allow to specify the parameter of the missing data mechanism, i.e., the rate of missingness. We let the domain expert define the probability of forgetting an event. The domain expert can specify how to weigh these probabilities against each other. That is, the expert can give preference to paths with higher probability (i.e., those paths that include transitions with high relative weights), or to paths with less missing events that are required to be inserted into the trace. This novel post-processing step on the cost-optimal alignments allows one to control the probability of paths in the model that are not reflected in a log by any event.

For example, consider a loop in a GDT_SPN model with $n$ activities in the loop. By setting the chance of missing entries low, e.g., setting the missingness probability to 0.1 (10 percent chance that an event is lost), an additional iteration through the loop will become more unlikely, as its path probability will be multiplied by the factor $0.1^n$. This factor is the probability that all $n$ events of an iteration are missing. We select the alignment with the highest path probability, and thereby determine the structure of the repaired trace. Then, we can continue and insert the times of the missing events (i.e., the identified *model moves* of the alignment) in the trace.

### 5.3.2 *Inference of the Time*

To insert the timing information, we need to combine the probabilistic information captured in the GDT_SPN model with the information that we have for each trace, i.e., the timestamps of the recorded events. Bayesian networks, as introduced in Section 2.5 on page 26, offer a solution for this task: Setting random variables to given values (that is, *inserting evidence*) and performing *inference* on the remaining variables.

A GDT_SPN model cannot be directly translated into a Bayesian network, because Bayesian networks do not support cycles. In the previous step, we identified the path through the GDT_SPN model that is optimal in terms of structural violations, and also most probable in terms of missing entries and chosen branches. With the path given, we can eliminate choices from the model by removing branches of the process model that were not taken. Thus, we *unfold* the net from the initial marking along the chosen path. Note that loops are but a special type of choices and are eliminated from the model by unfolding along a given trace. The unfolded GDT_SPN model can be converted into a Bayesian network, where we perform inference for the missing values, after inserting the observed evidence (i.e., occurrence time of observed events).

For example, reconsider trace $tr_1 = \langle A(0.0), B(1.6), D(1.9), C(4.2), B(4.46)\rangle$ and suppose that activity D was forgotten to be documented. That is, only trace

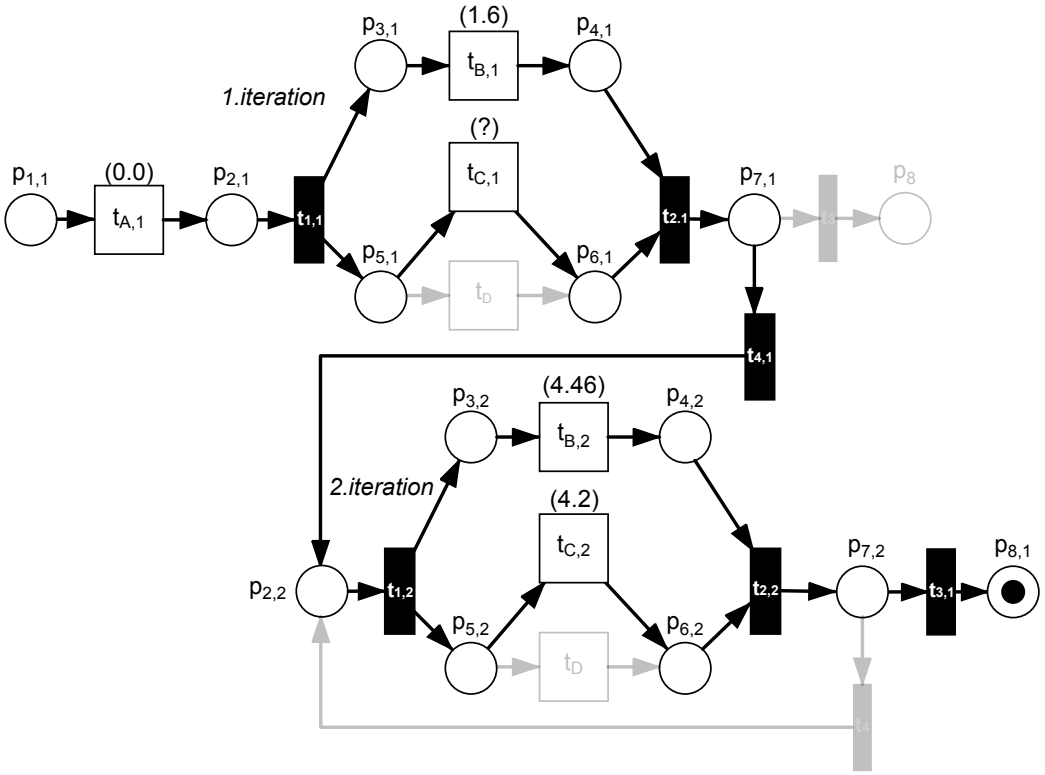



Figure 33: Unfolded model of trace $tr'_1$. The model is unfolded according to the alignment of trace $tr'_1$, cf. the similar case in Figure 16 on page 42. By unfolding the model according to the alignment, we get rid of certain paths and iterations that are allowed in the model, but are not selected in the path of the alignment.

$tr'_1 = \langle A(0.0), B(1.6), C(4.2), B(4.46)\rangle$ is observed. Then, we align $tr'_1$ to the running example model in Figure 7 on page 20. Suppose, we obtain the following alignment:

| *log* | A(0.0) | ≫ | ≫ | B(1.6) | ≫ | ≫ | ≫ | C(4.2) | B(4.46) | ≫ | ≫ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *model* | A | τ | C | B | τ | τ | τ | C | B | τ | τ |
| | $\mathbf{t}_A$ | $\mathbf{t}_1$ | $\mathbf{t}_C$ | $\mathbf{t}_B$ | $\mathbf{t}_2$ | $\mathbf{t}_4$ | $\mathbf{t}_1$ | $\mathbf{t}_C$ | $\mathbf{t}_B$ | $\mathbf{t}_2$ | $\mathbf{t}_3$ |

The unfolded model according to this alignment is shown in Figure 33. In the figure, the black parts mark the path taken in the model, while the gray parts have been removed during unfolding. Note that the unfolded model still contains parallelism, but it is acyclic. Thus, we can convert it into a Bayesian network with a similar structure, where the random variables represent timed transitions. Notice that due to multiple iterations of loops, activities can happen multiple times. We differentiate iterations by adding an occurrence index, e.g., $\mathbf{t}_{B,1}$ and $\mathbf{t}_{B,2}$ correspond to the first and second occurrence of the transition $\mathbf{t}_B$. The unfolding is done by traversing the model along the path dictated by the alignment and keeping track of the occurrences of the transitions and places.

We transform the unfolded model into a Bayesian network with a similar structure. Most immediate transitions are not needed in the Bayesian network, as

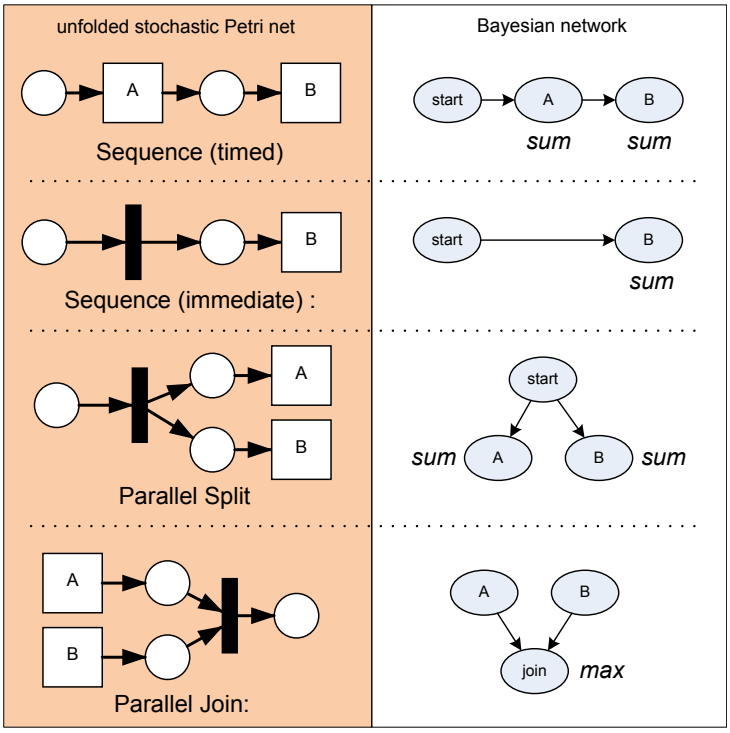



Figure 34: Transformation rules. The Transformation of unfolded GDT_SPN models to Bayesian networks.

these do not take time and no choices remain in the unfolded process. Only immediate transitions joining parallel branches are kept as nodes in the Bayesian network.

Figure 34 shows transformation patterns for sequences, parallel splits, and synchronizing joins. These are the only constructs remaining in the unfolded form of the GDT_SPN model. In the resulting Bayesian network, we use the *sum* and *max* relations to define the random variables given their parents. More concretely, if timed transition $\mathbf{t}_i$ is followed by timed transition $\mathbf{t}_j$ in a sequence, we can convert this fragment into a Bayesian network with random variables $X_i$ and $X_j$. From the GDT_SPN model, we use the transition duration distributions $\mathcal{D}(\mathbf{t}_i) = F_{\delta_i}$ and $\mathcal{D}(\mathbf{t}_j) = F_{\delta_j}$. Then, the parent variable $X_i$ has the unconditional probability distribution $P(X_i \leqslant t) = F_{\delta_i}(t)$ and the child variable $X_j$ has the conditional probability distribution function:

$$P(X_j \leqslant t \mid X_i) = \int_{-\infty}^{t} (f_{\delta_i} * f_{\delta_j})(t)\,dt = \int_{-\infty}^{t} \left( \int_{-\infty}^{\infty} f_{\delta_i}(x) f_{\delta_j}(t-x)\,dx \right) dt \quad (23)$$

The $*$ operator that we apply here is representing a *convolution* operation and is used for summation of independent random variables [36]. For each value of the parent $x_i \in X_i$, the probability distribution is defined as $P(X_j \leqslant t \mid X_i = x_i) = F_{\delta_j}(t - x_i)$, i.e., the distribution of $X_j$ is shifted by the value of its parent to the right. A parallel split, see lower left part in Figure 34, is treated as two sequences sharing the same parent node.

The *max* relation that is required for joining branches at synchronization points, cf. the lower right pattern in Figure 34, is defined as follows: Let $X_i$ and $X_j$ be the parents of $X_k$, such that $X_k$ is the maximum of its parents. Then,

$$P(X_k \leqslant t \mid X_i, X_j) = P(\mathit{max}(X_i, X_j) \leqslant t) = P(X_i \leqslant t) \cdot P(X_j \leqslant t) = F_{\delta_i}(t) \cdot F_{\delta_j}(t)$$

that is, the probability distribution functions are multiplied.

In Bayesian networks that we construct this way, inference is an NP-hard problem [52]. To increase the performance and the applicability of our approach, we convert this generic Bayesian network model into a linear Gaussian model where inference can be done efficiently in $\mathcal{O}(n^3)$, where $n$ is the number of nodes [113], see Section 2.5. Therefore, each distribution is approximated with a normal distribution capturing the first two moments (i.e., the mean and the variance) of the distribution. Then, we need to handle the *max* relation, which is no linear combination of its parents. Note that the maximum of two normally distributed random variables is not normally distributed. Therefore, we use a linear approximation, as described in [216]. This means that we express the maximum as a normal distribution, with its parameters depending linearly on the normal distributions of the joined branches. The approximation is good, when the standard deviations of the joined distributions are similar, and the approximation degrades when they differ, cf. the experiments by Zhang et al. in [216].

Once we constructed the Bayesian network, we set the values for the observed events for their corresponding random variables, i.e., we insert the evidence into the network. Then, we perform inference in the form of querying the posterior probability distributions of the unobserved variables. For latter task, we use the Bayes Net Toolbox (BNT) for Matlab [141], which implements the inference methods. This corresponds to the second step in the *insert time* part of Figure 32.

The posterior probability distributions of the queried variables reflect the resulting probabilities, when the conditions are set according to the evidence. Our aim is to get the *most likely* time values for the missing events. These most likely times are good estimators for the time when the events occurred in reality, and thus can be used by process participants as clues during root cause analysis. For example, in order to find the responsible person for the task in question, an estimation of when it happened *most likely* can be helpful. Then, obvious documentation errors can be corrected by the responsible person.

Note that repaired values with most likely time values need to be treated with caution, as they do not capture the uncertainty of the missing values. Therefore, we mark repaired entries in the event log as *artificially inserted*. We envision a monitoring environment, where these inserted event entries are passed as suggestions to the corresponding process participants to correct potential documentation errors.

Once we determined the most probable values for the timestamps of all missing events in a trace, we proceed with the next trace in the log starting another iteration of the algorithm. In the next section, we evaluate the quality of the events that we artificially inserted into the traces.

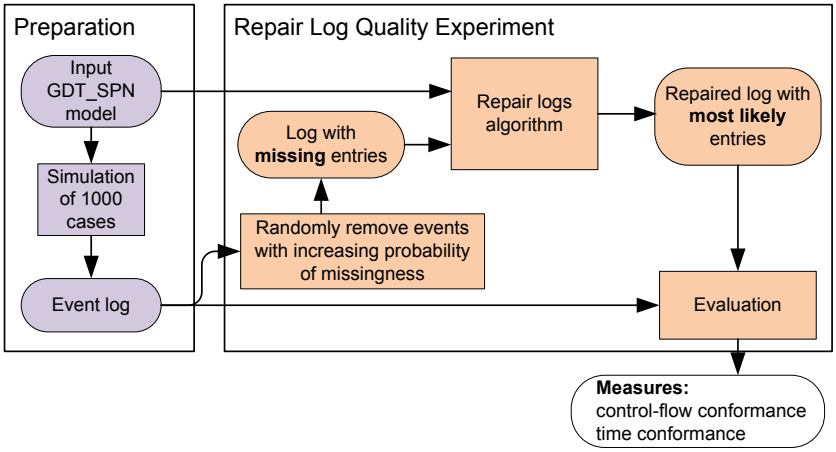



Figure 35: Setup to evaluate repair quality. We assume that the GDT_SPN model captures the behavior of the process correctly. Therefore, we simulate traces according to the model and collect them in an event log. We randomly remove events (maintaining at least one event per trace), and repair them according to the model. We evaluate how well the repaired entries match the original entries.

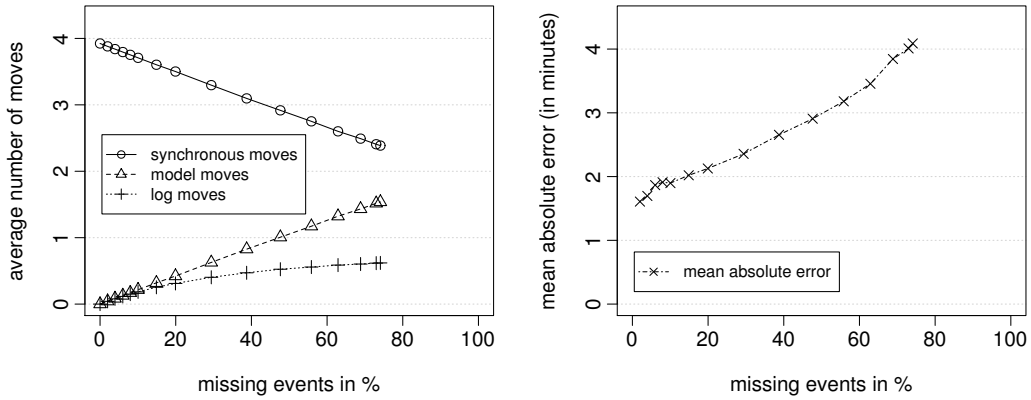
## 5.4 CONCEPTUAL EVALUATION

We evaluate the structural correctness of the repaired traces, and the error of the time values of the repaired events compared to the time values of the original events.

### 5.4.1 *Experimental Setup*

To evaluate the quality of the algorithm, we follow the experimental setup described in Figure 35. The assumption of the normative model allows us to skip the cross-validation part. We assume for the conceptual evaluation that we know the GDT_SPN model that captures the process characteristics. We simulate 1000 traces based on the GDT_SPN model to get an event log. Then, the simulated event log is passed to the experiment. Here, we simulate a missingness process by randomly removing events from the event log. The chance of removing an event is increased with each iteration, but we ensure that at least one event remains in every trace. We pass the event log that contains missing entries with the GDT_SPN model to the repair algorithm that we described in Section 5.3.

The repair algorithm's output is then compared with the original traces to see how well we could restore the missing events. We use two measures for assessing the quality of the repaired log. The cost-based *fitness* measure, cf. Section 2.3, compares how well a model fits a log. It is based on the costs of asynchronous movements. For this evaluation, we want to distinguish model moves and log moves. Therefore, we use the raw metrics of the number of synchronous moves, model moves, and log moves. This way, we have a measure to check, whether we repaired the right events in the right order. For measuring the quality of repaired timestamps, we use a simple measure by comparing the real event's time with




(a) Number of synchronous, model, and log moves depending on the number of missing events from the log.

(b) The mean absolute error of the repaired times (in minutes).

Figure 36: Results for the running example. Evaluation results for repairing 1000 traces of the model in Figure 17 with an increasing amount of noise on the x-axis.

the repaired event's time. This makes sense if we have chosen the correct event to be inserted. We use the mean absolute error (MAE) of the events that have been inserted. This is the arithmetic mean of the absolute differences between repaired event times and original event times. Formally, let $E_{rem}$ be the events that are removed from an event log. Let $e_o \in E_{rem}$ with $\gamma(e_o) = t_o$ be the original event that we removed from a trace and let $e_i$ be the repaired event with its time $\gamma(e_i) = t_i$. Then the error of the repaired event's time is $err(e_o) = t_i - t_o$. The MAE of all such events is

$$\text{MAE} = \frac{\sum_{e_o \in E_{rem}} |err(e_o)|}{|E_{rem}|}.$$

### 5.4.2 *Imputation Results and Interpretation*

We first evaluate the repair algorithm's ability to repair a trace correctly with the GDT_SPN model introduced in Figure 17 on page 45.

#### *Repair Results of the Running Example*

The experiment was performed with a log of 1000 simulated traces. Figure 36 displays the (a) structural, and (b) time quality which we achieved by repairing an event log with randomly removed events. Each dot is based on the repair results of this log with a different percentage of randomly removed events. In Figure 36a, the raw fitness metrics of the alignment are shown. The solid line with circles shows the number of *synchronous moves*. The other two lines are the number of model moves (dashed line with triangles) and the number of log moves (dotted line with crosses) necessary to align the two traces.

We cannot guarantee the ordering of events due to parallelism in the model. A change in the ordering of two events in the repaired trace results in a *synchronous move* for one event, and a *log move* and a *model move* for the other event—to

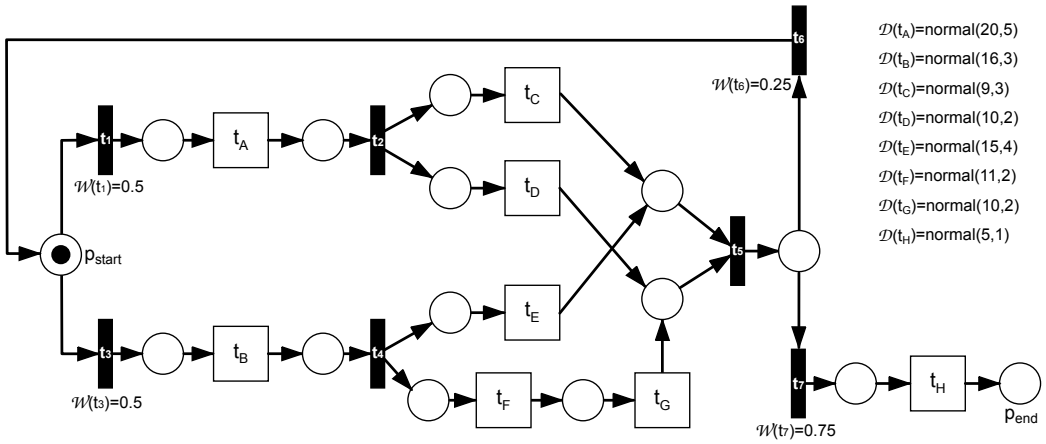



Figure 37: Non-free choice example GDT_SPN model. This model has normally distributed durations for its timed activities. There are more dependencies between activities as in our running example. When omitted, weights of immediate transitions are 1 by convention. Labels for places are omitted, too, except for the initial place $p_{start}$ (with a token) and for the final place $p_{end}$.

remove it from one position and insert it in another. Currently, the algorithm uses time distributions (i.e., the posterior probability distributions of missing events) to determine the time of the missing event and thereby the ordering of multiple parallel activities. Consider an extreme case, where $n$ identically distributed transitions are started in parallel. The chance to restore the correct ordering (if no further information exists) is $\frac{1}{n!}$ in this case.

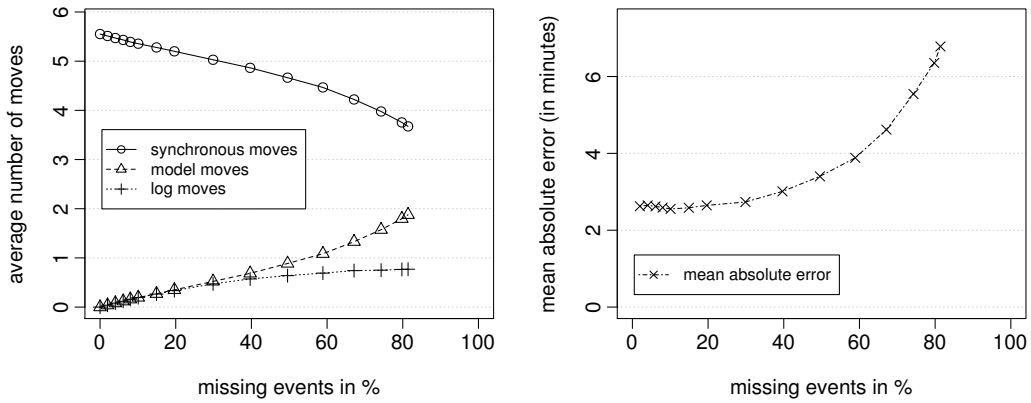
In Figure 36b, we can see the mean absolute error in the relative time units specified in the model. The graph shows that the offset between original event's time and repaired event's time increases with the amount of noise. In this case the relationship looks like a linear one. This is not necessarily the case, however, as we will see in the next example.

*Results for a Second Example*

Because our running example is mostly meant to illustrate the concepts of the performance model used throughout this thesis, cf. Definition 4 on page 16, we also evaluate the repair quality of the proposed algorithm in a more complex model.

In Figure 37, another GDT_SPN model is depicted. We use it as a second example for the repair algorithm. The activities are normally distributed with most of the probability mass in the positive area. In this example, there exist more dependencies between activities, i.e., more activities share the *sequential* relationship. Thus, we expect to be able to correctly repair more events than in the previous example with comparable amount of noise.

The results for the second example model (cf. Figure 37) are depicted in Figure 38. As expected, the quality of the repaired model does not suffer that much from missing events at low noise levels, because the algorithm is able to restore missing events that are in a sequential relationship to observed events. Note




(a) Number of synchronous, model, and log moves depending on the number of missing events from the log.

(b) The mean absolute error of the repaired times (in minutes).

Figure 38: Results for non-free choice example model. Evaluation results for repairing a simulated log of the GDT_SPN model in Figure 37.

that a process model with only sequential relationships exhibits only one possible trace, which can be always correctly repaired, when at least one event is observed.

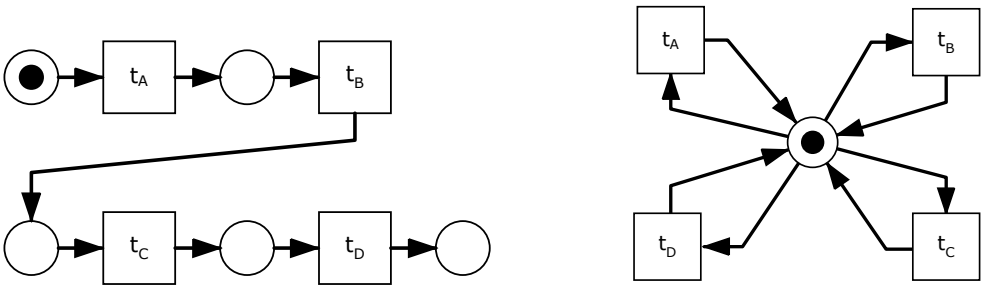
It is interesting to note that for this second example, the absolute error of the repaired times exhibits a non-linear relationship to the number of missing events in the log. There are two reasons for this non-linear growth of error, compared to the previous results for the running example. First, the distributions of the times in the first example are a mix of uniform, normal and deterministic times (the first lognormal distribution is not taken into account, as the first event in the log marks the completion of activity A, and we do not know the start). In contrast, the second example only contains normally distributed values.

A second reason is the non-linear decrease of synchronous moves with increasing noise in the second model. It is intuitively clear that if we do not restore the events in correct order, the time of the missing events will also become less accurate.

## 5.5 DISCUSSION

To summarize the evaluation of the synthetic models, we note the following observations:

MODEL GENERALITY The quality of the structural repair is influenced by the generality of the model, i.e., by the possible behavior that the model allows. Obviously, a constrained model that allows only one sequence (see Figure 39a) of execution is easy to restore structurally, as opposed to an ad-hoc process, as for instance a flower model (see Figure 39b), where the execution sequence of activities is arbitrary, and correct restoration is impossible. In our experience, the generality of most business process models




(a) Sequence. (b) Flower model.

Figure 39: Model Generality. In comparison two models with transitions $\mathbf{t}_A, \ldots, \mathbf{t}_D$ are depicted. The left model allows exactly one execution trace $\langle A, B, C, D \rangle$, while the flower model on the right allows to execute the activities in any order, any number of times.

is between these two extremes, and sequential dependencies between activities are common.

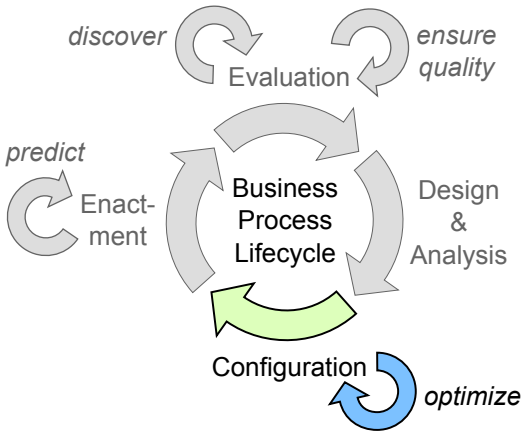
TIME QUALITY The quality of the repair of the timestamp of the missing events depends on multiple factors. First, to better estimate the times of the activities, it helps to have selected the correct alignment, that is, the time quality depends on the quality of the structural repair. Second, the underlying distributions of the durations play a decisive role in the quality of the repaired times. The best achievable estimate is bound by the variance of the duration distributions. More precisely, the estimates are bound by the variance of the *conditional* distributions—conditioned on the timestamps of the observed events before and after an activity.

OPTIONAL BRANCHES When more than one activity exists on optional branches or in loops, moderate amounts of noise can be repaired structurally. In this case, the quality is non-linear. With higher probability of missingness, the probability increases that whole branches of the process are missing and thus cannot be repaired by the algorithm. In contrast, missing entries for single optional events cannot be repaired by the algorithm, leading to a rather linear relationship between noise and repaired quality, in case of a model with many optional single activities.

With these insights, we now turn to another problem that arises in manual process execution environments. The next chapter deals with the question, at which points it makes most sense to document a process.

# 6 SELECTION OF MONITORING POINTS

*This chapter is based on the results published in [172].*




Having dealt with errors in documentation in the previous chapter, we now also consider that manual documentation takes time and is costly. Therefore, we want to identify the most relevant points, when documentation is required to produce optimal prediction results.

In terms of the business process lifecycle, we provide a method to assist *configuration* of a process. This method can also be employed to identify the optimal selection of event monitoring points, when configuring a monitoring architecture, cf. Section 2.4.

### CHAPTER OUTLINE

This chapter is structured as follows. First, we introduce the problem and describe the setting of monitoring in manual process environments in more detail. We compare related work in Section 6.1 and present an approach to the problem in Section 6.2. We evaluate how well the approach performs conceptually in Section 6.3. Finally, in Section 6.4, we discuss our approach and its limitations.

## INTRODUCTION

When asked, physicians often complain about the burden of documentation. In fact, in a survey in Maryland in 2007, more than half of the nurses reported that they spend 25–50 percent of their shift with completing patient documentation [82]. In the following, we concentrate on the use of documentation for monitoring purposes.

One straightforward way to reduce time and cost is to avoid unnecessary documentation. Obviously, if certain documentation steps are required due to legal reasons [27] or for accounting purposes, we cannot simply discard them. The remaining documentation steps, however, should be examined more closely, and we are interested in identifying the most relevant documentation steps. We measure the relevance of a certain documentation step in its effect on the monitoring and prediction quality of the process.

*Business process monitoring* assists in performance estimation. For example, it is required for the prediction of time until completion of a process instance or the duration of certain activities, see Chapter 4 and our earlier work on probabilistic monitoring [176]. Monitoring enables the detection of performance issues, e.g., being behind schedule, so that corrective actions can be taken to finish the affected instances as planned, and to avoid deviations from target goals and service level agreements.

Recall that, in this thesis, we assume that a monitoring architecture, cf. Section 2.4, is used in an organization. A monitoring architecture collects information (e.g., manual documentation, logs) that is produced by process participants and systems for monitoring and analysis purposes. Although there is ongoing work on the topic of (semi-)automatic configuration of such architectures, cf. [29, 158, 157], an automatic mapping between raw event sources and business processes is not yet usable in production. Therefore, a high amount of *manual* work is required to wire business processes with raw event sources.

By the terminology that we use in this thesis, we are binding event that reflect state changes from the IT landscape of an organization to event monitoring points (EMPs), cf. Definition 6 on page 24. Recall that EMPs mark state changes in the lifecycle of activities in a business process. Besides deciding, which tasks process participants should document, we motivate the work in this chapter also with the problem of manually wiring events in the process environment to EMPs. We would like to have a means to decide, *which events* we should monitor to ensure optimal prediction quality during the execution of business process instances. Thereby, we want to keep the number of these configurations low, because each additional installed EMP costs time and money.

## 6.1 RELATED WORK

A related problem can be found in the research field of *project management*. There, the optimal timing of control points within a project is important. Control points are moments in a project where control activities are conducted to compare the measured project state against the project plan. On the one hand, control activ-

ities are required, because they allow the detection of project deviations from the planned schedule and the implementation of corrective actions. On the other hand, they produce direct costs, are time-consuming, and bind resources. Therefore, the decisions to be made include:

- How many control points are required during a project?
- At which intervals should the project progress be controlled?

Partovi and Burton [152] evaluate the effectiveness of five control timing policies in a simulation study: equal intervals, front loading, end loading, random and no control. These policies refer to the density of control points in a project. It turns out that the heterogeneity of the projects, rendered comparison of the policies infeasible. De Falco and Macchiaroli argue that individual allocation of monitoring activities is required [67]. To that end, they provide a method to quantitatively determine the optimal control point distribution by first defining the effort function of a project based on activity intensity and slack time. This knowledge (i.e., to know the phases of high activity) allows one to place control points more densely in accordance with the activity.

Raz and Erel also rely on the notion of activity intensities [163]. They determine the optimal timing of control activities by maximizing the amount of information generated by each single control point. The amount of gathered information is calculated based on the intensity of activities carried out since the last control point and the time elapsed since their execution. Raz and Erel solve the optimization problem by a dynamic programming approach. It seems promising to apply the same technique to the optimal allocation of EMPs in a process model as well. In contrast to control points that can be distributed continuously in a project, EMPs can only be positioned at well-defined places in a process model, e.g., at the end of activities.

Another related use case for optimal allocation of control points is the capability to diagnose systems. In this setting, a diagnosis component of a system (e.g., a microprocessor) tries to identify reasons for unexpected behavior [48]. A set of sensors (observations) and a model of the system is required to detect system components that are responsible for incorrect behavior. Installed sensors divide the system into clusters. The connections between components inside of a cluster are not monitored, only the connections with components outside of a cluster can be observed. Ceballos et al. [48] present a concept to allocate a set of new sensors to maximize the system diagnosability. Their approach maximizes the number of monitored clusters with a given number of additional sensors. To tackle the exponential complexity of this maximization problem, the authors resort to a greedy algorithm. Their greedy algorithm considers bottlenecks in the system as candidates for allocating one sensor at a time. This approach is transferred to business processes in the work of Borrego et al. [40, 41]. They install control points within a process to identify the activities that are responsible for deviating behavior from the process model. For the allocation, the authors rely on the algorithm in [48]. Latter algorithm focuses on increasing the number of monitored activity clusters. However, a maximum number of activity clusters does not necessarily yield an optimal prediction quality.

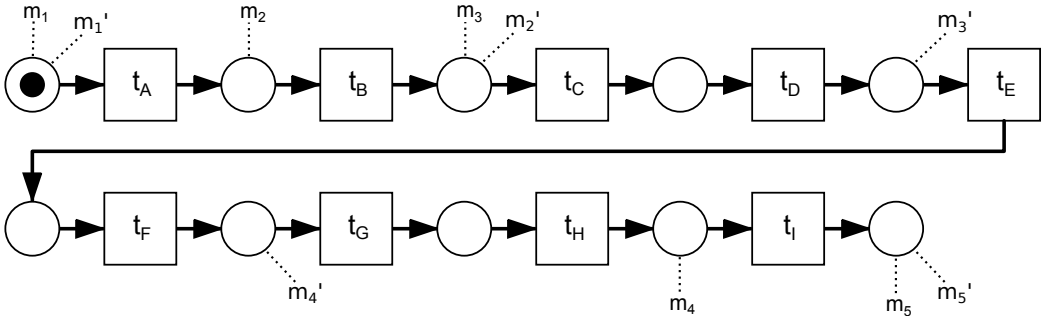



Figure 40: Process model with differently allocated EMPs. In the first scenario $(m_1, \ldots, m_5)$ the EMPs are concentrated at the beginning and the end of the process, while in the second scenario $(m'_1, \ldots, m'_5)$, the EMPs are allocated rather uniformly throughout the process.

## 6.2 OPTIMIZATION OF THE SELECTION

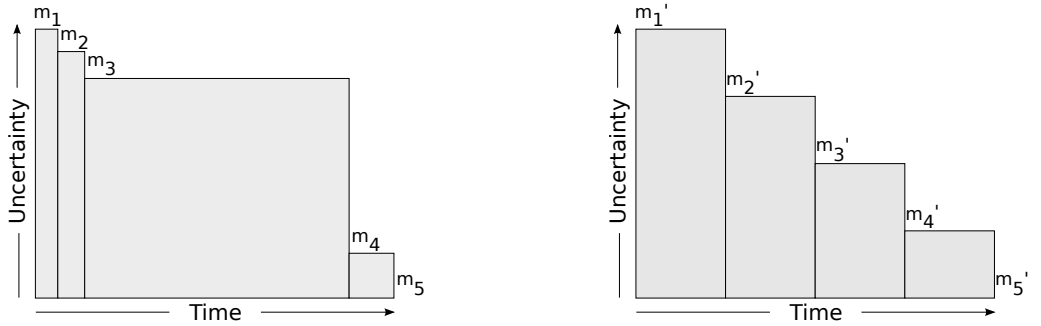
In the following, we assume that an organization wants to reduce cost by only configuring the set of event monitoring points in a monitoring architecture that provides the highest prediction quality over the process duration. In other words, the objective is to keep the uncertainty of the remaining process duration as low as possible over the entire process duration. Let us illustrate this by an example. Consider the sequential process in Figure 40.

In the figure, two configurations of EMPs are annotated: in the first scenario, we measure the start ($m_1$), the end of activity A ($m_2$), the end of activity B ($m_3$), the end of activity H ($m_4$), and the end of the last activity I ($m_5$). In the second scenario, the second measurement is after activity B ($m'_2$), the third after D ($m'_3$), and the fourth after F ($m'_4$).

Let us assume that the activities in this process are all similar—both in terms of mean duration that they require for completion, as well as in the variance in their duration. Then, we can sketch the uncertainty of the remaining process duration as a function over time that is highest when we start an instance, and decreases when we get more information about the progress of the process instance. Finally, when the instance completes, the uncertainty about the remaining process duration becomes zero.

We can depict the uncertainties in a graph by using the elapsed mean time as the x-axis, and the uncertainty of the remaining process duration as the y-axis. Figure 41 illustrates the effect of the choice of EMPs on the uncertainty of the process duration. Note the big block in Figure 41a, which indicates a long period of high uncertainty in the process. A more uniformly distributed selection of EMPs yields a more balanced amount of uncertainty during the execution of the process, cf. the optimal allocation in Figure 41b.

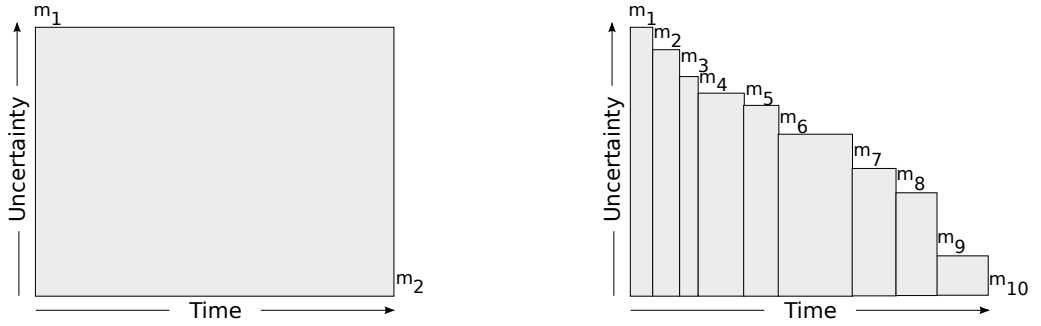
The previous example showed the intuition of selecting a set of EMPs for monitoring. That is, it showed the effects of *which* EMPs are selected. Another question that appears in this context, is *how many* EMPs we want to install. Or—translated to real life settings—how many activities should be documented by




(a) Inhomogeneous monitoring (b) Homogeneous monitoring (optimal)

Figure 41: Inhomogeneous and homogeneous monitoring allocation. The overall uncertainty of the process duration is depicted for the process in Figure 40. The (a) first scenario allocates event monitoring points $m_1, \ldots, m_5$, and the (b) second scenario allocates $m'_1, \ldots, m'_5$. We consider the configuration with the smallest uncertainty area under the graph as optimal.

the process participants (or alternatively, how many EMPs should be configured in a monitoring architecture).




(a) Minimal monitoring effort. (b) Maximal monitoring effort.

Figure 42: Minimal and maximal monitoring allocation. Distribution of uncertainty in dependence of the number of EMPs. The width of a column is the mean duration between the EMPs, the height is a measure of uncertainty, e.g., mean square error. In (a), only start and end of a process instance is measured. In (b), all EMPs are set up to reach the highest possible prediction quality.

In Figure 42, we consider two extreme cases. That is, we either only monitor the start and the end of the process, as in Figure 42a, or we monitor the start of the process and the end of every activity, as depicted in Figure 42b.

With this intuition of the problem, we can now proceed to the formulation of the optimization problem that we solve in this chapter.

### 6.2.1 *Problem Formulation*

For this chapter, we restrict process models to consist of sequential activities only. For each process, we require that it has a start event and an end event that can

be monitored. The end event coincides with the termination of the last activity in the process. Let us assume that the termination event of each activity can be monitored, i.e., we can connect the termination EMP of each activity to an event source from the process execution environment. Then, for each monitoring point, we are interested in the remaining mean duration:

**Definition 15 (Mean Remaining Duration)** *The mean remaining duration of a process is a function* $mean_{dur}\colon EMP \to \mathbb{R}_0^+$ *assigning to each event monitoring point* $m_i \in EMP$ *the arithmetic mean of the durations from the time at* $m_i$ *until the termination time of the process which is captured by the termination of the last activity in event monitoring point* $m_{(n+1)}$.

The remaining mean duration is strictly decreasing from activity to activity in sequential processes, as we do not consider activities with negative durations, i.e., for all pairs of monitoring points $(m_i, m_j)$ (with $i < j \leqslant n+1$, where $n$ is the number of sequential activities) it holds that $mean_{dur}(m_i) \geqslant mean_{dur}(m_j)$.

**Definition 16 (Uncertainty of the Remaining Duration)** *The uncertainty of the remaining duration is a function* $u_{dur}\colon EMP \to \mathbb{R}_0^+$ *assigning a non-negative value to an EMP. The function* $u_{dur}$ *captures an uncertainty measure of the remaining process duration.*

We allow the domain expert to select an appropriate uncertainty function that captures the uncertainty of the remaining duration at EMPs. An example measure is the sample variance, i.e., the spread of individual process durations from the mean process duration at an EMP. More sophisticated measures of uncertainty, as the RMSE that we used in Chapter 4, can also be computed, by simulating a prediction at each EMP with a cross-validation of the event log. For the method that we describe, it is only necessary that we can quantify the uncertainty of the remaining process duration.

**Definition 17 (Monitoring Uncertainty)** *Let* GDT_SPN *be a sequential process model with the set of transitions* $T = \{t_1, \ldots, t_n\}$ *and the corresponding set of possible event monitoring points* $EMP = \{m_1, \ldots, m_{n+1}\}$, *where* $m_1$ *denotes the start of the process, and each* $m_i, 1 < i \leqslant n+1$ *denotes the firing of transition* $t_{i-1}$, *i.e., the end of the corresponding activity in the process. The overall uncertainty* $U$ *in the process is defined as:*

$$U(m_1, \ldots, m_{n+1}) = \sum_{i=1}^{n} u_{dur}(m_i) \cdot (mean_{dur}(m_i) - mean_{dur}(m_{i+1}))$$

Note that, with this definition, we interpret the monitoring uncertainty as the area under the stair-shaped uncertainty figures, cf. Figure 41, and Figure 42. We formulate the problem of selecting the optimal event monitoring points (EMPs) as follows:

PROBLEM: Find a subset of size $k$ of the available event monitoring points, such that the monitoring uncertainty $U$ using that subset is minimal.

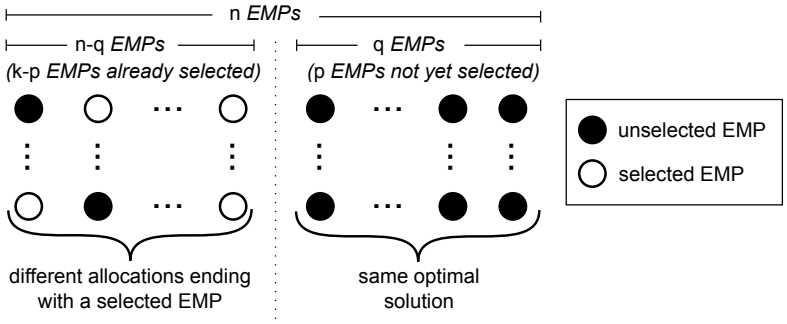



Figure 43: Different combinations with same local optimal solutions. When choosing $k$ out of $n$ potential EMPs, we can partition the EMPs into two parts, and look at the sub-problem to choose $p$ EMPs ($p < k$) out of the remaining $q$ EMPs. Then, as long as the last EMP $m_{n-q}$ is selected in the first part, the combinations of $k-p$ selected EMPs in the first $n-q$ potential EMPs are irrelevant for the remaining $q$ EMPs. That is, their configuration does not affect the remaining monitoring points, as latter only depend on the last EMP in the chain.

### 6.2.2 *Approach and Algorithm*

Now that we have formalized the optimization problem, we can formulate the solution algorithm for optimal placement of EMPs in a process model, such that the overall uncertainty is minimal. The algorithm is inspired by the approach in [163] that was proposed for the allocation of control points in projects. Notice that in contrast to the approach in [163], where control points can be allocated in arbitrary intervals (e.g., on a daily basis) in projects, the selection of EMPs is only possible at well-defined positions, that is, at state changes of an activity. Therefore, the maximum amount of EMPs is limited by the combinations of activities in a process model and the state transitions of activities. In the examples we have seen so far in this theses, we only encountered a single event per activity that denoted the termination of the activity. This is but a coincidence, and can be extended to more complex activity lifecycles [206]. For example, if we assume an activity lifecycle with two transitions (events are mappable to the start and the end of an activity), the maximum number of EMPs is $2n$ (with $n$ being the number of activities in the sequential process model).

The problem of selecting $k$ EMPs optimally out of $n$ potential EMPs is computationally complex. There exist $\binom{n}{k}$ combinations to consider for a solution. Fortunately, we can make use of the fact that a local optimal solution, i.e., which $p$ EMPs to select (with $p < k$) from $q$ subsequent potential EMPs (with $p < q < n$), only depends on the last selected EMP, and not on the entire combination of all previously selected EMPs.

Figure 43 illustrates the property that for a remaining assignment of $p$ out of $q$ EMPs, the combinations of the previous EMPs do not matter. More specifically, there exist $\binom{n-q-1}{k-p-1}$ combinations in this example for the previous $n-q$ EMPs (depicted as different allocations on the left hand side of Figure 43). All these combinations share the same local optimal solution for the remaining $p$ out of $q$

EMPs. This makes a dynamic programming approach, as also proposed in [163], feasible.

In optimization problems, it is possible to transform a problem into its *dual*, that is a complementary form. We make use of this possibility, and instead of minimizing the overall uncertainty $U$, we maximize the reduction of uncertainty of a given number of selected EMPs. Note that in a sequential process model, the allocation of an additional EMP $m_j$ reduces the overall uncertainty of the prediction by the uncertainty of the activities since the last EMP $m_i$. This reduction applies for the remaining mean duration $mean_{dur}(m_j)$. The reduction of uncertainty $\bar{U}$ can be interpreted as the white area that complements the stair-shaped uncertainty area to a rectangle that is as high as the highest uncertainty at the start and as wide as the mean process duration of the process. Thus, we define the overall reduction of the uncertainty $\bar{U}$ of the allocated EMPs as:

$$\bar{U}(m_1, \ldots, m_{n+1}) = \sum_{i=2}^{n} (u_{dur}(m_{i-1}) - u_{dur}(m_i)) \cdot mean_{dur}(m_i) \tag{24}$$

Note that this sum starts at index $i = 2$, because we require to monitor at least the start and end of the process. In this term, each individual summand represents the reduced uncertainty for the remainder of the process. Let $m_i, m_j$ be subsequently selected EMPs with $i < j$. We define $\bar{U}(m_i, m_j)$ as the removed uncertainty by selecting $m_j$:

$$\bar{U}(m_i, m_j) = (u_{dur}(m_i) - u_{dur}(m_j)) \cdot mean_{dur}(m_j) \tag{25}$$

We are interested in the maximal reduction of uncertainty that we can achieve by adding an EMP $m_j$ after the last EMP $m_i$. We call this quantity $\bar{U}_1^*(m_i)$:

$$\bar{U}_1^*(m_i) = \max_{i<j\leqslant n} (\bar{U}(m_i, m_j)) \tag{26}$$

However, we are even more interested in the EMP (i.e., the argument) that maximizes the reduced uncertainty:

$$m_{j_1}^* | m_i = \arg\max_{i<j\leqslant n} (\bar{U}(m_i, m_j)) \tag{27}$$

With these equations, we mathematically captured the optimal solution for adding *one* additional EMP. The problem, we try to solve is more complex, as we are also interested in allocating two or more EMPs, given the last EMP. We denote the multiplicity with an index. That is, $\bar{U}_2^*(m_i)$ expresses the maximum reduction of uncertainty with two additional EMPs after the last EMP $m_i$:

$$\bar{U}_2^*(m_i) = \max_{i<j\leqslant n-1} (\bar{U}(m_i, m_j) + \bar{U}_1^*(m_i)) \tag{28}$$

With this, we can also recursively define the more general formula for the reduction of overall uncertainty by an additional k EMPs:

$$\bar{U}^*_k(m_i) = \max_{i<j\leqslant n-(k-1)} (\bar{U}(m_i, m_j) + \bar{U}^*_{k-1}(m_j)) \tag{29}$$

Again, we are interested in the arguments that yield the optimal reduction of uncertainty, i.e., the selection of EMPs that maximize this formula:

$$m^*_{j_k}|m_i = \arg\max_{i<k\leqslant n-(k-1)} (\bar{U}(m_i, m_j) + \bar{U}^*_{k-1}(m_j)) \tag{30}$$

With this notation, we can reformulate the problem, cf. Section 6.2.1, as to compute $\bar{U}^*_{(k-2)}(m_1)$ (cf. Equation 29) (i.e., the maximum reduction of uncertainty gained by k event monitoring points, given that two of them measure the start and the end of the process), and return the arguments $m^*_{j_{(k-2)}}, m^*_{j_{(k-3)}}, \ldots, m^*_{j_1}$, cf. Equation (30). To solve this problem, the proposed algorithm pursues the following steps [172]:

1. Determine the set *EMP* of potential EMPs in the given process model. Let $n = |EMP|$ be the number of EMPs.
2. For each $m_1, \ldots, m_n \in EMP$ calculate the remaining mean duration $mean_{dur}(m_i)$ until process termination based on given historical execution data. We can rely on the technique presented in Chapter 3 and compute the mean of the discovered distributions of the activities.
3. Calculate the uncertainty $u_{dur}(m_i)$ of the remaining durations for each potential EMP based on historical execution data according to the given uncertainty function (or from the learned distributions).
4. Compute $\bar{U}^*_{(k-2)}(m_1)$ for the given number k of requested EMPs by using dynamic programming for searching through the $\binom{n-2}{k-2}$ solution combinations. Thereby, intermediate computed optima are stored to save time by not recomputing such solutions. Note that by relying on dynamic programming we pay with computer memory to improve computing speed.

With a small extension, the presented algorithm can be also used to determine the required number of EMPs to meet a given uncertainty threshold. Therefore, the algorithm has to be executed iteratively by incrementing the number k of allocated EMPs, starting with 2, until the threshold is met. If the given threshold cannot be met, even with all potential EMPs selected, this extended algorithm returns the overall uncertainty inherent to the process to the user.

## 6.3 CONCEPTUAL EVALUATION

In the previous chapters, we conceptually evaluated the quality of the methods. In this chapter, however, we face an optimization problem, for which we find

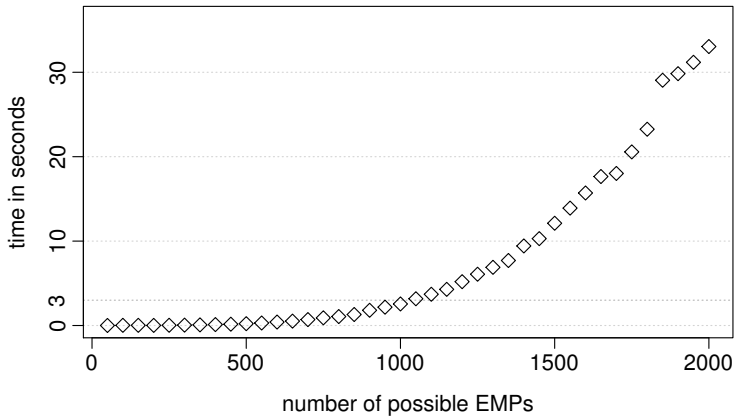



Figure 44: Selection algorithm runtime performance. The plot shows the durations in seconds to find the optimal subset of EMPs in relation to the problem size. The parameter $k$ is set to 10 percent of the number of possible EMPs. Up to selecting 100 event monitoring points out of 1000, the runtime of the algorithm is under 2 seconds. Above the number of 1800 possible candidate EMPs, computer memory shortage causes irregularities.

the optimal solution by enumerating all possible candidates. That is, we are *guaranteed* to find the optimal subset of EMPs that yields the minimal uncertainty of the remaining duration over time. Therefore, we only conduct a scalability analysis, to see, at which model sizes the limits of the dynamic programming approach are.

We tested the scalability of the algorithm in a controlled experiment. Therefore, we randomly generated stair shaped uncertainty graphs with $n = 50, 100, \ldots, 2000$ nodes and computed the optimal selection of $k = \frac{n}{10}$ EMPs in each run with a laptop computer. Figure 44 depicts the results, that we also presented in [172]. Notably, determining the optimal allocation of 100 EMPs out of 1000 potential candidate EMPs takes less than three seconds, cf. Figure 44. The number of possible combinations of allocation is $\binom{1000}{100} = 6.4 \cdot 10^{139}$. In this case, dividing the problem into independent sub-problems, creates $3.9 \cdot 10^7$ optimization steps, of which only 87 495 need to be computed, and the rest can be retrieved from the dynamic programming table (i.e., cached result). Most business process models that we encountered contain around 10-100 activities, which the algorithm could handle very fast.

We shall investigate the effect and potential of finding the optimal allocation of event monitoring points in the next part of this thesis, where we apply this method in a case study with real data from a hospital surgery process.

## 6.4 DISCUSSION

We have seen that, conceptually, the algorithm can deal with large process models, but we need to point out the limitations of the current method and potential future improvements. The major current limitations of the approach are:

INDEPENDENCE OF ACTIVITIES Correlation between activities is not taken into account explicitly.

SEQUENTIAL PROCESSES The presented algorithm is requiring sequential processes at the moment. To a certain degree processes with decisions or cycles could also be handled by this approach, if a single path in the model is much more frequently taken than others.

Dependencies between activities are cumbersome, as they can cause the uncertainty of the remaining process duration to rise from one EMP to the next EMP. While in the motivation, we sketched the shape of the uncertainty graph as a decreasing stair-shaped function, cf. Figure 41, it can occur that after an activity that is negatively correlated with a subsequent activity finishes, the uncertainty in the remaining process duration is higher after the second measurement than after the first. Assume the extreme case that there are only two sequential activities in a process that are negatively correlated with a correlation coefficient of $-1$. That is, the durations of activities $A$ and $B$ are in a linear relationship, such as $\alpha A + \beta B = \text{constant}$. Then, the variance of $A + B$ can be zero (in the case that $\alpha = \beta$), compared to the variance of the duration of the activity $B$, which is greater than $0$.

Such an extreme case could happen, if the process always took a fixed amount of time, regardless of when the work was completed. In this case, measuring the end of the first activity is indeed useless for estimating the remaining process duration. It is enough in this case to know when the process started.

Although the algorithm does not explicitly support correlation information on the uncertainty of the remaining durations, it does support the latter case of negative correlation implicitly. That is, it would not select an EMP between two negatively correlated activities, if the uncertainty of the second activity was greater than the uncertainty of both activities combined.

The second limitation (i.e., the restriction to sequential process models) is due to complex dependencies of EMPs in parallel branches of a process model. We discussed some of the difficulties in [176], i.e., that it is possible to gain knowledge about a parallel branch based on the occurrence of an event that is monitored in another parallel branch in the process. Figure 45 shows that the influence of a single event monitoring point on a branch of a process extends to other parallel branches. Notice that this effect is related to the flow of dependencies in Bayesian networks, where paths of influence (i.e., sequences of consecutive edges) exist in the graph and can be blocked by setting certain variables to a value. See the discussion on paths of influence in probabilistic models in Pearl's book [155, Section 1.2.3].

An approximation of the solution to more complex processes structures would be possible by numerical analysis, as proposed van der Aalst et al. in [6], and detailed in the thesis of Reijers [166], and similarly by Eder and Pichler in [65].

We also want to mention the following minor limitations:

SIMPLIFIED MODEL The uncertainty function assigns one value to each EMP. However, as we have seen in Chapter 4, the distribution function of an activity conditioned on elapsed time is changing, as time proceeds—unless

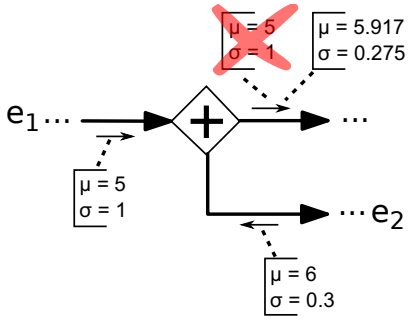



Figure 45: Effect of an event on the distribution of a parallel branch. Upon the detection of event $e_2$ on a parallel branch, the estimation on the start time of the upper branch gets updated from the original normal distribution $N(5, 1^2)$ to the more accurate conditional distribution $N(5.917, 0.275^2)$, cf. [176].

the duration is exponentially distributed. The uncertainty can decrease, or even increase as time passes, which we did not consider in the optimization problem.

MODEL ELICITATION If we used the optimized selection approach to select which event monitoring points should be connected in a monitoring architecture, we would require performance models to optimize the selection. We would not have the performance data, however, and could use expert estimates, or it would be possible to manually sample duration of process activities.

Although there are some limitations to the approach, we provide means to select the event monitoring points (EMPs) for an optimal prediction quality. The mathematical formulation allows to use any uncertainty function $u_{dur}$ in the computation. An obvious extension is to assign weights to EMPs that can be added to the optimization function (cf. Equation 24) by prioritizing specific EMPs. For example, differing setup costs could be taken into account this way, too, without increasing the complexity of the problem or the algorithm.

# Part III

# APPLICATION, EVALUATION & DISCUSSIONS

# 7 IMPLEMENTATION IN PROM

We implemented the methods and algorithms that we proposed in the previous part in ProM[1] (see also the report on ProM by van Dongen et al. [62]).

## CHAPTER OUTLINE

This chapter is dedicated to the implementation and it also highlights design decisions. Readers who are not interested in implementation aspects can skip this chapter and continue reading the application of the concepts to real use cases in Chapter 8. Here, we first give an introduction to ProM in Section 7.1. Then, we describe the technical details of the GDT_SPN model and the extension in the Petri Net Markup Language (PNML) format in Section 7.2. Finally, we describe the implemented plug-ins and the architecture in Section 7.3.

1 The ProM framework http://www.promtools.org

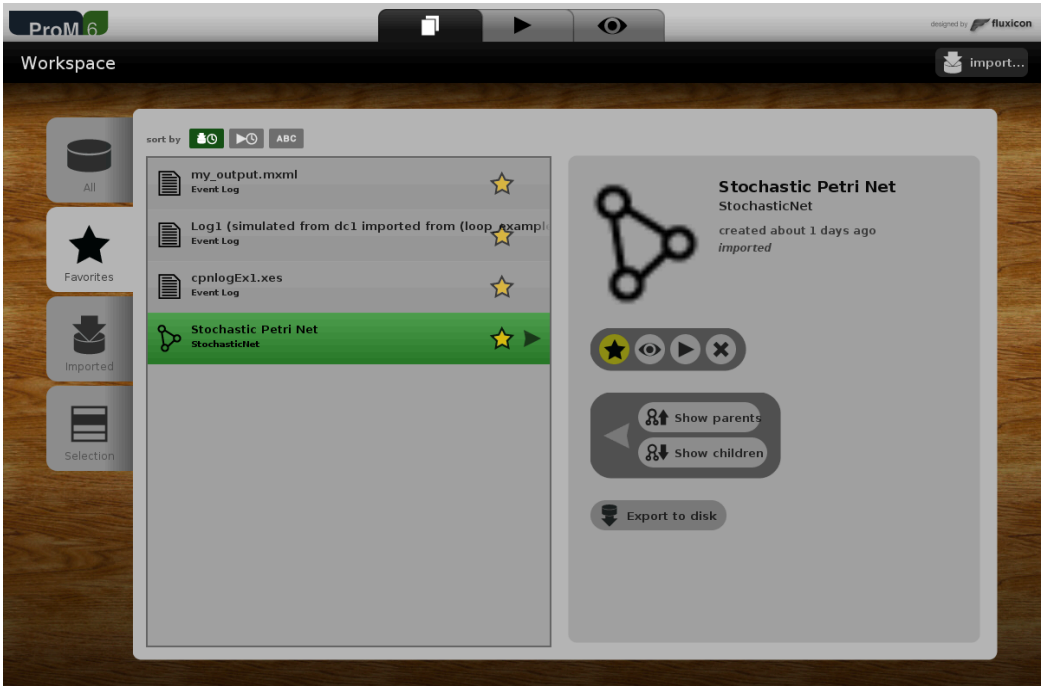



Figure 46: Screenshot of the user interface of ProM. Here, several objects are loaded in the object pool, i.e., a GDT_SPN model (selected), and three event logs. The available objects can be processed with plug-ins, e.g., we could mine a Petri net model from an event log.

## 7.1 INTRODUCTION TO PROM

ProM is an open-source framework for collecting tools and applications of process mining [62]. The framework is written in the programming language Java [80] and is extensible by packages. These packages can contain plug-ins that are included by the framework and provided to the user for execution.

The ProM framework is actively maintained, such that it matured to a professional level [202]. Some of its highlights include the following [202]:

- clear separation of plug-in functionality and the graphical user interface
- a common object pool that serves as input and output to plug-ins
- plug-in variants for *overloading* of plug-ins to handle different input sets

Beside these mentioned features, the fact that it is maintained as free software helped to make ProM a successful toolkit. By the time of writing this thesis, the ProM website lists more than 500 plug-ins in more than 120 packages[2] available.

An architectural description of the ProM framework is not in the scope of this thesis. Instead, we refer to the documentation available in [202]. Figure 46 shows the user interface of ProM with the available objects that can be analyzed with the installed plug-ins.

Next, we describe the extensions in ProM that implement the concepts that we introduced in the previous parts.

2 `http://www.promtools.org/prom6/` (last accessed on April 25, 2014)

## 7.2 PNML EXCHANGE FORMAT

First, we need to capture the extensions to Petri nets, which we support, i.e., GDT_SPN specific attributes, such as priority, weight and distribution of transitions. We did not have to create the entire GDT_SPN model specification from scratch, as Petri nets are already supported in the ProM framework—in fact, the very first discovery algorithms generate Petri nets from event logs [12].

As standardized format for exchanging Petri nets between tools the Petri Net Markup Language (PNML) was developed [37]. The specification provides an extension mechanism, which can be used for adding specific properties. For our work, we made the design decision to use the available extension mechanism, instead of waiting for the promised third version of the PNML specification.

It turned out that with only a few extensions to the Petri net model, we are able to capture the entire GDT_SPN model. Recall, that GDT_SPN models are Petri nets with additional properties for transitions, i.e., priority, weight, type (immediate or timed), and distribution of the duration.

We define an extension in the `toolspecific` element that can be added to all elements in the PNML specification [37]. The extension is kept minimal. That is, for each property that we require for transitions in GDT_SPN models, we add a key-value pair.

This is best illustrated with an example. Suppose we want to serialize a timed transition $\mathbf{t}_A$ with weight $\mathcal{W}(\mathbf{t}_A) = 0.5$, with priority $\mathcal{P}(\mathbf{t}_A) = 0$, with a normal distribution with mean $\mu = 12$ and standard deviation $\sigma = 2.7$. Then, the transition's representation in PNML can be written as:

```
<transition id="tA">
  <name>
    <text>A</text>
  </name>
  <graphics>
    <!-- omitted -->
  </graphics>
  <toolspecific tool="StochasticPetriNet" version="0.1">
    <property key="priority">0</property>
    <property key="weight">0.5</property>
    <property key="distributionType">NORMAL</property>
    <property key="distributionParameters">12;2.7</property>
  </toolspecific>
</transition>
```

The PNML standard allows to extend the Petri net model in arbitrary fashion with the `toolspecific` element. The only requirement is that the name of the tool and a version number is given [37]. Our extension is currently using the `StochasticPetriNet` name and a preliminary version number of 0.1. We assess this extension as preliminary proof-of-concept, and extensions (e.g., raw data, censored data, time units) might be added to the model in the future.

Notice the four properties that we added to the transition through this extension. The keys are predefined, but self-explanatory, e.g., `distributionType` for the type of distribution. The key and value for priority and weight of a transition

are added as integer, and float values respectively. The distribution is specified by both a type (`distributionType`) and parameters (`distributionParameters`). Table 1 lists the distribution types that are currently supported for transitions in GDT_SPN models:

Table 1: Supported distribution types and their PNML keys.

| distribution type | value in PNML | parameters |
|---|---|---|
| Immediate | `IMMEDIATE` | 0 |
| Normal | `NORMAL` | 2 |
| Log-normal | `LOGNORMAL` | 2 |
| Exponential | `EXPONENTIAL` | 1 |
| Beta | `BETA` | 2 |
| Gamma | `GAMMA` | 2 |
| Uniform | `UNIFORM` | 2 |
| Student-T | `STUDENT_T` | 2 |
| Weibull | `WEIBULL` | 2 |
| Gaussian kernel density | `GAUSSIAN_KERNEL` | unbounded |
| Histogram | `HISTOGRAM` | unbounded |
| Log-spline density | `LOGSPLINE` | unbounded |
| Deterministic | `DETERMINISTIC` | 1 |

The number of parameters for the distributions vary, if more than one parameter is required, the values are separated with semi-colons (`;`). For example, the immediate transition does not require any parameters, because it takes by definition always 0 time. Another example is the deterministic distribution, which only requires one parameter specifying the time point, when it usually fires. For non-parametric distributions (e.g., histograms, log-spline distribution), we store the raw data in place of parameters, i.e., the sample points to which a non-parametric distribution can be fit.

With these four extensions for transitions in regular Petri nets, we are able to capture the full GDT_SPN model specification, as given in Definition 4. We will now introduce the plug-ins that we developed for the presented conceptual methods of Part ii.

## 7.3 IMPLEMENTED PLUG-INS

To be able to work with the GDT_SPN model, we extended the ProM Petri net classes by subclasses that capture the additional stochastic information. First, we need plug-ins to be able to read and write this format.

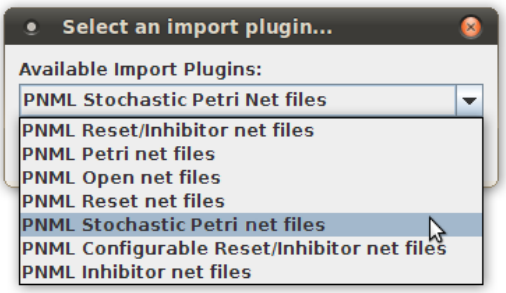



Figure 47: Import plug-in for stochastic PNML models.

*Import and Export of Stochastic Petri Nets*

We realized the import and export with the eXtensible Markup Language (XML) serialization framework Simple[3] that can be configured by annotating the classes and attributes and provides serialization to and de-serialization from XML. The respective plug-ins are registered in ProM and the user can select the format `StochasticPetriNet` when opening a PNML file. When a GDT_SPN model is obtained from historical data, this plug-in can export the model to the PNML format with the extensions described in Section 7.2.

We also want to be able to import previously exported (or manually created) GDT_SPN models from PNML files. Figure 47 shows the choices for available Petri net formats in ProM. The selected entry *PNML Stochastic Petri net files* marks the import plug-in that can be used to de-serialize GDT_SPN models.

*Enriching Petri Nets with Stochastic Information in ProM*

We implemented the enrichment algorithm (cf. Section 3.4) in ProM as a plug-in taking two arguments: (1) an event log and (2) the Petri net model. The two arguments are aligned to each other by the cost-based alignment technique that is presented in [15], and also implemented in ProM in the `PNetReplayer` package. Once aligned, the collection of stochastic information can start. Therefore, the user needs to specify a distribution type, the time unit to be used in the enriched Petri net (e.g., whether the mean of a normal distribution is expressed in minutes, hours, or days), and the execution policy, cf. Section 3.3, which the model shall obey.

We use standard statistical techniques to fit simple distributions to the data. For the non-parametric case that is able to deal with censored data, we rely on the log-spline algorithm by Kooperberg [114], which is implemented in the statistics software R[4]. The output of the plug-in is a GDT_SPN model, which can not only be used for gaining insights into the process (i.e., identify bottlenecks, comparing frequency of paths), but also for monitoring and predicting termination time of process instances, or reasoning about missing documentation.

3 Simple – XML serialization: `http://simple.sourceforge.net/`
4 The R Project for Statistical Computing: `http://www.r-project.org/`

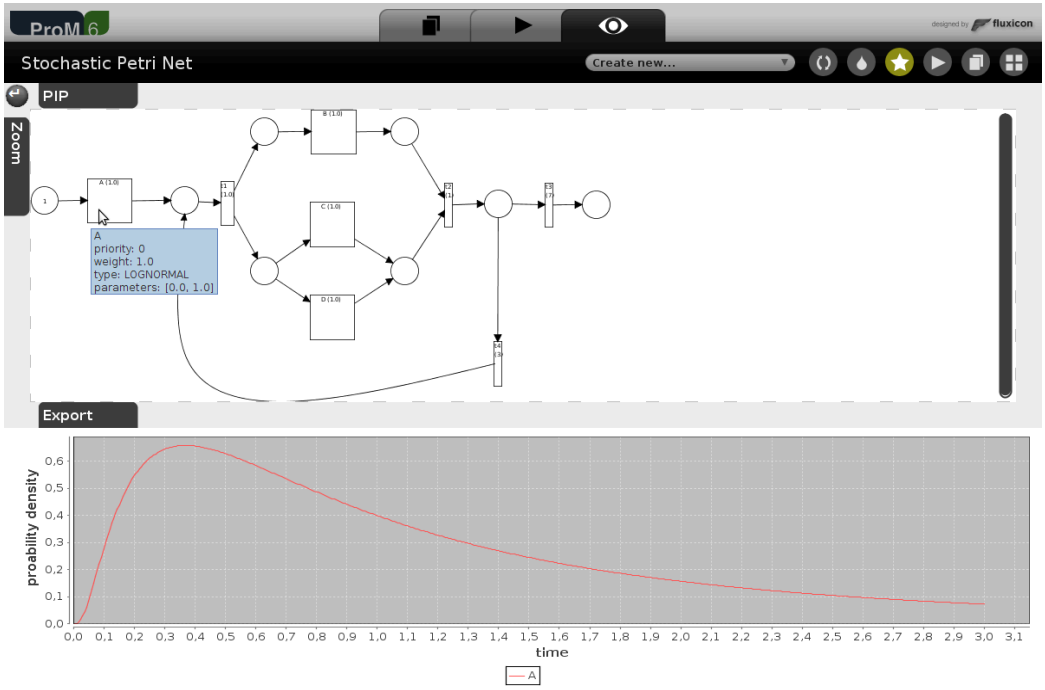



Figure 48: An obtained model with stochastic information. The probability density function of the selected timed transition is visualized in a frame below the model.

In Figure 48, we can see the plug-in that visualizes the additional stochastic information stored in a GDT_SPN model. The graphical representation of the probability density function of the currently selected transition is realized using the `JFreeChart` library[5].

*Prediction of Remaining Durations in ProM*

At the point when we extended the Petri net model in ProM, there were already several semantics implemented for different flavors of Petri nets. The extension to generally distributed transition stochastic Petri net (GDT_SPN) models turned out to be simple. However, after some experimentation, we found out that the implementation could be optimized, e.g., finding the enabled transitions of a marking in a bounded Petri net can be done in constant time, by precomputing the reachable markings.

Functionality to determine which transition is enabled in the simulation, as well as to take random samples of the probability distributions of the timed transitions, can be implemented efficiently. As described in Section 4.2.2, we need to be able to draw random samples from the truncated distributions of timed distributions for simulating the remaining process duration. The simplest way to draw random samples from truncated distributions is to do rejection sampling. Rejection sampling draws a sample from the unconditional distribution, then checks whether it fits the condition, and either accepts it as valid sample, or rejects it, if it does not match the condition, and repeats this process until a

5 JFreeChart: `http://www.jfree.org/jfreechart/`

valid sample is drawn. This technique is very useful for distributions, for which efficient sampling techniques are known in the general case. If the truncation is extreme, i.e., most of the probability mass is truncated, this technique suffers from inefficiency, as the probability to obtain valid samples can near 0. For that case, we resort to the following method.

*Slice sampling* is a very efficient sampling method to sample from arbitrary distributions [149]. It only requires that we are able to compute the density of a probability distribution. Note that we provided the formula of truncated distributions in Section 4.2.2. Slice sampling works by performing a random walk in the two-dimensional area under the density function. In slice sampling, uniform sampling alternates in horizontal and vertical slices of this area. The samples stay within the boundary of the density function. The values of the horizontal coordinates are collected after each iteration, and they converge to the distribution described by the density function. Neal has shown latter property for density functions $f$, where $f(x) > 0$ for all $x$, cf. the original article in [149].

However, convergence is not guaranteed for all cases. For instance, it is possible to design a very extreme probability distribution, which has two modes that are far away from each other, and in between the probability is zero. Then, the random walk under the density function will have no chance to reach the second mode in practical implementations of the algorithm. Note that such theoretical extreme cases are irrelevant for durations in practical business processes. Usually, if there is a chance that an activity takes much longer than usual, there should also be a chance that it finishes at some point in time between the usual and that long duration.

Note that when dealing with probability values, zero probabilities should be avoided or used with great caution, as a probability value of 0 stands for impossibility. For our case of modeling durations, it is reasonable to assume that a duration cannot be negative, but to exclude intervals between probable durations is not a sensible thing to do.

Another property that we need to be aware of, when relying on slice sampling, is that subsequent samples in a sequence of samples are correlated. This unwanted effect is negligible for unimodal distributions, but can be observed for multimodal distributions, where transitioning from one mode to another is unlikely. We can mitigate the effect in any case, by taking a sufficiently large number of samples (i.e., enough samples such that the distribution is approximated), and shuffling the samples randomly before returning them.

We can conclude that although in theory there might be some issues with slice sampling, the technique offers a very efficient way to produce samples in practical use cases. We will evaluate the efficiency of the sampling methods in Chapter 8.

*Repairing Event Logs with Missing Entries*

For brevity, we only highlight the design decisions made for implementing the method to reason about the missing events in Chapter 5. The input to the algorithm is an event log and a GDT_SPN model.

Again we use the alignment method to choose the path through the model for an individual trace. The second step (i.e., performing inference to gain insight into the unobserved parts in the trace) is done by converting the GDT_SPN model into a Bayesian network. Note that before conversion, we get rid of cycles by unfolding according to the alignment, as described in Section 5.3.2. We then convert the model to a Bayesian network that we build in the Bayes Net Toolbox for Matlab (BNT) [141]. If the duration distributions are not normally distributed, however, we approximate them with normal ones by computing the first two moments, i.e., the mean and the variance of the distributions. The resulting Bayesian network is a linear Gaussian model, and the inference is done by the (BNT) implementation. We collect the results and store both the most likely timestamp, as well as the marginal distributions for events that are missing according to the model in the log.

The inference procedure is a computationally intensive task. We use Octave[6], a free Matlab-clone, for running BNT [141]. To increase performance of the algorithm, we support parallelism by splitting the inference to multiple worker threads. The user can select how many worker threads to create, and can thereby utilize additional processor cores.

*Optimally Selecting Event Monitoring Points*

Finally, we also implemented the algorithm for optimally selecting EMPs in order to reduce uncertainty. We build the selection on the existing `TransitionSystems` plugin [201]. The uncertainty function $u_{dur}$ can be selected to be an error measure (e.g., the mean absolute error, or the root mean square error) from a cross-validated prediction experiment, or simply based on the sample variance of the remaining duration at each EMP.

Figure 49 shows the user interface to the selection of monitoring points. The visualization presents the uncertainty graph that decreases over time as more information gets available at the EMPs. The user can select the desired number of EMPs on the top left of the figure. The maximal monitoring effort (i.e., when using all EMPs for prediction) overlays the optimal allocation given the desired number of EMPs. This allows the user to compare the relative uncertainties of the selections. The optimal selection of the possible EMPs are highlighted with an arrow and bold type. The uncertainty function can be selected to be either computed directly from the samples (e.g., to use the sample variance), or by a simulated prediction experiment using cross validation (e.g., to compute the RMSE of the predictions at each EMP).

With the implementation of the concepts, we evaluate the contributions of this thesis with case studies in the next chapter.

6 GNU Octave `http://www.gnu.org/software/octave/`


Figure 49: Uncertainty visualization and optimal selection of EMPs. In the figure three EMPs are selected, that is, the algorithm computed the optimal position of $m_2$, while $m_1$, and $m_3$ are fixed at the start and end of the process. As uncertainty function $u_{dur}$, the sample variance of the remaining durations is selected. The overall uncertainty area is normalized by division of the maximal uncertainty value and the overall mean process duration to an area of 1. This way, the overall uncertainty of the selection of EMPs can be compared to the setting of selecting all event monitoring points.

# 8

# EVALUATION WITH CASE STUDIES

In the previous chapters, we laid the foundations and introduced the concepts of discovery, prediction, imputation and cost-efficient allocation of measurement points. Here, we evaluate the applicability of these concepts for real life settings. Fortunately, in the course of this dissertation, we got access to process data of a hospital, a logistics provider, and an insurance company. We use this data to check, whether we can still observe the improvements encountered in the conceptual presentation.

## CHAPTER OUTLINE

This chapter is organized as follows. First, we introduce the industry use cases with which we evaluate our approaches in Section 8.1. To evaluate the discovery method, we measure the bias between discovered GDT_SPN models and real event logs in Section 8.2. Then, to evaluate the prediction method, we compare our approach with state of the art work on prediction and benchmark methods in Section 8.3. Further, in Section 8.4, we evaluate the repair approach with the use cases. Last, we evaluate the optimization of monitoring points in Section 8.5.

## 8.1 INDUSTRY USE CASES

In this section, we introduce the processes that are used for the evaluation of the concepts in real settings.

*Surgery Process in a Clinic*

Recall that we are motivated by the settings encountered in hospitals to work with probabilistic models in this thesis. Therefore, the first use case that we investigate is the surgery process of a hospital that comprises treatment relevant steps. Efficient management of the surgery process is decisive for hospitals, because the operating room is usually their most costly asset [124].

We use data from a Dutch university clinic, which we also analyzed in [110]. The process is depicted as a Petri net model in Figure 50. It is a rather sequential process that deals with both outpatients and ordered inpatients. Each transition corresponds to a treatment step that a nurse records in a spread sheet with timestamps. In the process, the patient is either ordered from the ward, or arrives through the emergency department. Latter option is not recorded in the documentation that we obtained, resulting in the missing transition on the lower branch. Later at some point, the patient arrives in the lock to be prepared for the surgery. Once the operating room (OR) is ready, the patient leaves the lock and enters the OR. In the OR, the anesthesia team starts the induction of the anesthesia. After that, the patient optionally gets an antibiotics treatment. The surgery starts with the incision (i.e., the first cut with the scalpel) and finishes with the suture (i.e., the closure of the tissue with stitches). Next, the anesthesia team performs the emergence from the anesthesia, which ends when the patient

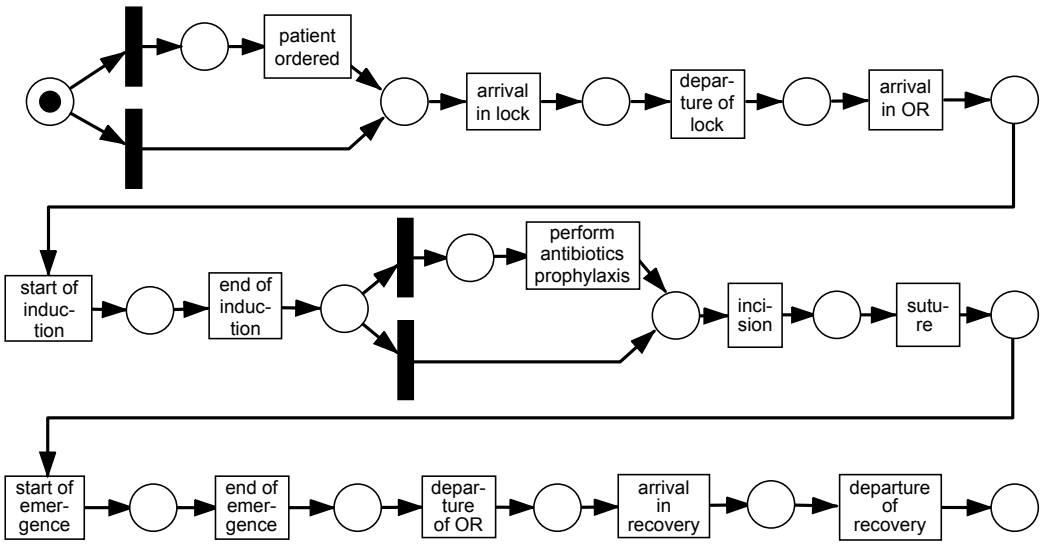



Figure 50: Surgery model. The Petri net model for a surgical procedure in a Dutch hospital.

has regained consciousness. Finally, the patient leaves the OR and is transported to the recovery.

In the corresponding event log 1310 cases are recorded. According to the model in Figure 50, there are twelve mandatory events and two optional events that can be observed. As we have investigated [110], there exist some conformance issues, i.e., missing events for some of the mandatory events, as well as swaps in the order of the tasks. Consider the antibiotics treatment that is optional in the model and only allowed after the induction to anaesthesia and before the incision. In the event log, out of 295 cases, where antibiotics treatment was documented, only 183 happened at this exact position in the process [110]. This kind of non-conformance often appears with real data that is not recorded by a process execution engine, and proves to be an additional challenge for the conceptual methods we introduced in Part ii. Later in this chapter, we shall see how strong these non-conformance issues affect our approach.

*Shipment Import Process of a Logistics Provider*

To demonstrate the more general applicability of our methods to other domains, we also evaluate the methods on a process that we encountered at a logistics provider in the Netherlands.

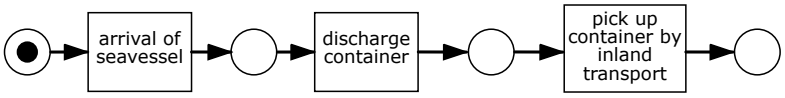



Figure 51: Logistics process model. The process model captures three sequential stages of a logistics process: the arrival of seavessels, container discharges, and further inland shipping.

Figure 51 depicts the model of the import process. Each case in the event log reflects a container. The corresponding event log contains event entries for each of the transitions in the model. First, a sea vessel arrives, then, the container is discharged, until it finally is picked up by inland transport to reach its destination in Europe. In this example, the conformance between log and model is very good because of the high degree of automation and standardization. In fact, the *fitness* between model and log is 1, that is, the log fits the model perfectly.

*Loan Application Process of a Financial Institute*

As a third process, we analyze the loan application process of a Dutch financial institute. The event log contains 262 000 events in 13 087 cases. In contrast to the previous two event logs, which we cannot disclose, this event log is publicly available [60]. The event log is very detailed and comprises three different related processes. We only consider the top-level process that is depicted in Figure 52. See also the analysis by Adriansyah and Buijs [14].

In Figure 52, we see a Petri net model that captures the top-level process of the loan application. On this level, we find the typical stages of a successful

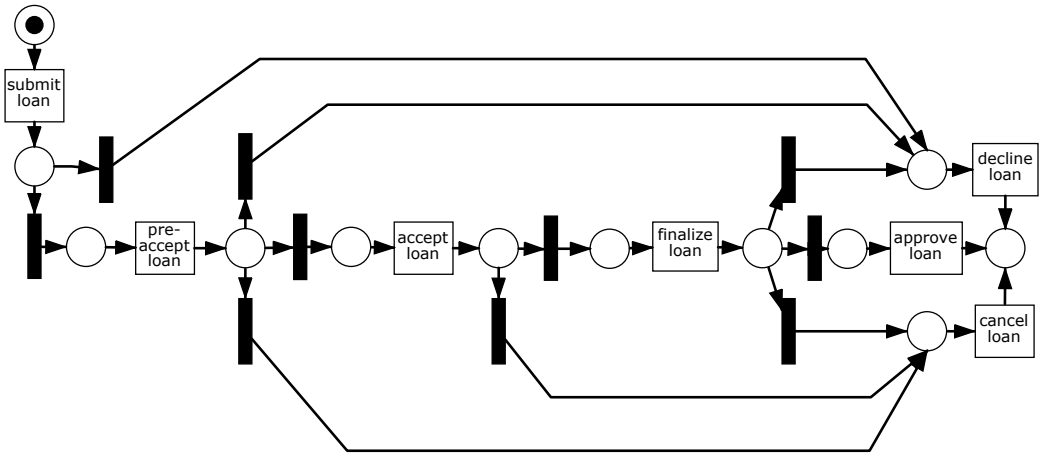


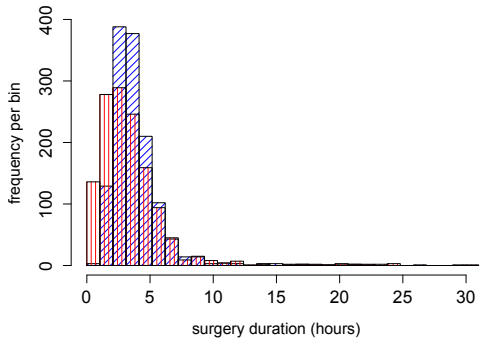

Figure 52: Loan application process model. This Petri net process model captures a high level view on a loan application process. Besides the sequential *happy path* (i.e., submitting, preaccepting, accepting, finalizing, and approving the loan), there is also the option for the client to cancel the loan or for the clerk to decline it.

loan application, i.e., first the loan is submitted, then it gets preaccepted, then accepted, and after finalization it is approved. Beside this regular path, the client can cancel the loan application, or the clerk can decline it at several stages. Notice that not all choices are present in the model, e.g., the clerk cannot accept and immediately reject the loan afterwards. Although this might be possible in reality, the model is based on the observable behavior of the process, where such a case did not occur.

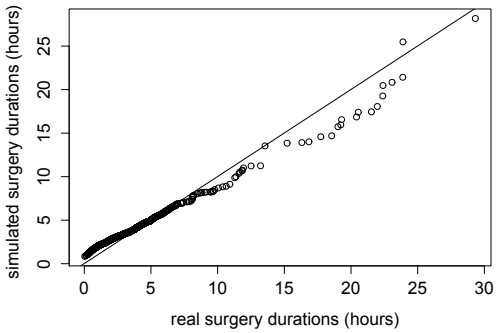
## 8.2 OBTAINED PROCESS MODELS

When we introduced the method to *obtain* a generally distributed transition stochastic Petri net (GDT_SPN) model from an event log and a matching Petri net model in Chapter 3, we looked at the introduced bias of the technique on an artificial process model. Therefore, we compared the accuracy of the obtained distribution of individual timed transitions and the accuracy of the transition weights at decision points. We already saw the effects of log size (i.e., the number of samples) on the learned model parameters.

In this section, we evaluate how well we can learn the behavior of a process, i.e., how well the obtained model reflects the overall process. To achieve this, we perform the following experiment. We first enrich Petri net models for each of the use cases in Section 8.1, and obtain GDT_SPN models. We use latter to simulate traces of execution of the process model and compare the traces with the original ones. When we have collected enough sample traces, we can calculate simple statistical measures, e.g., the sample means, but we can also compare the two distributions by statistical tests, e.g., the Kolmogorov-Smirnov test. This way, we can test how well the learned model is able to reproduce the observed behavior.




(a) Comparison of histograms of observed real durations (red vertical stripes) and simulated durations (blue diagonal stripes).




(b) Quantile-quantile plot for the real surgery durations and the simulated durations.

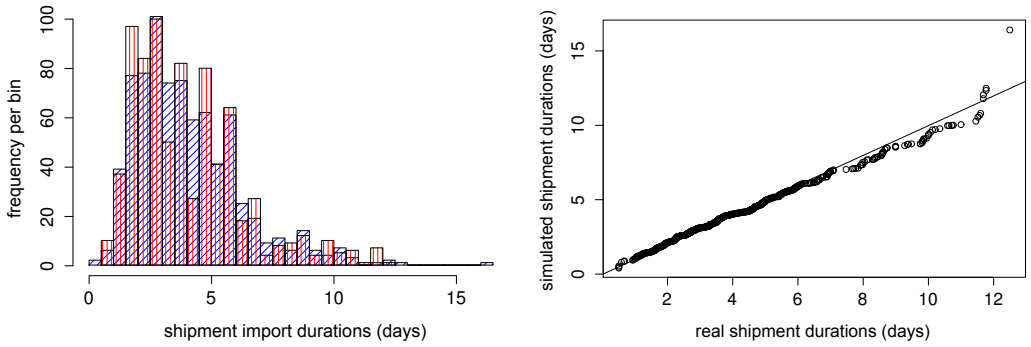
Figure 53: Surgery process model results. The simulated results are compared with the observed results. We see that much of the probability mass overlap. The shorter surgery duration values, however, are underrepresented in the simulated traces.

*Obtained Model Quality for the Surgery Process*

To obtain GDT_SPN models, as described in Chapter 3, we need to choose certain parameters of the model, that is, the selection policy and the distribution family of the timed transitions. For the surgery process model, see Figure 50, the choice of selection policy has no effect, because it contains no parallelism or loops. For estimation of the timed transitions we rely on the work of Kooperberg [114] that uses log-spline density estimators. In that approach, the overfitting problem is mitigated by the provided implementation—The implementation uses the Akaike information criterion (AIC) [20] that penalizes models with too many degrees of freedom. Thus, a certain balance between overfitting and generality of the learned distribution is provided.

To compare with the real 1310 traces, we simulated an equal number of artificial traces. Figure 53 shows two alternative visualizations to compare the real traces with the simulated traces from the obtained model. We can interpret the histogram representation as follows: the simulated (blue diagonal stripes) traces contain less cases with short durations up to two hours, but more cases that take two to five hours. The histogram of the simulated durations is shifted to the right and is also narrower than the histogram of the original durations (red vertical stripes), which shows the bias that we have between the model and the real traces.

There are several potential explanations for the model bias. The most important one is the quality of the event log. As we discussed in [110], there are missing entries in this manually documented event log, such that only 540 traces out of 1310 fit the model exactly. For example, there are traces where only the ordering of the patient and the arrival in the lock is documented, see Figure 50, which results in underestimation of the *real* surgery duration in the real traces. Our obtained GDT_SPN model assumes that no event is forgotten and that a simulation always yields full traces that fit to the model.




(a) Comparison of histograms of observed real durations (red vertical stripes) and simulated durations (blue diagonal stripes).

(b) Quantile-quantile plot for the real surgery durations and the simulated durations.

Figure 54: Logistics shipment process results. .

Besides the effect of missing data on the learned model, another source of bias is that our obtained GDT_SPN model is not able to capture dependencies between activities. For example, it might be that for simple surgeries each step is short, or all activity durations tend to take longer due to other causes, e.g., too many patients waiting in parallel.

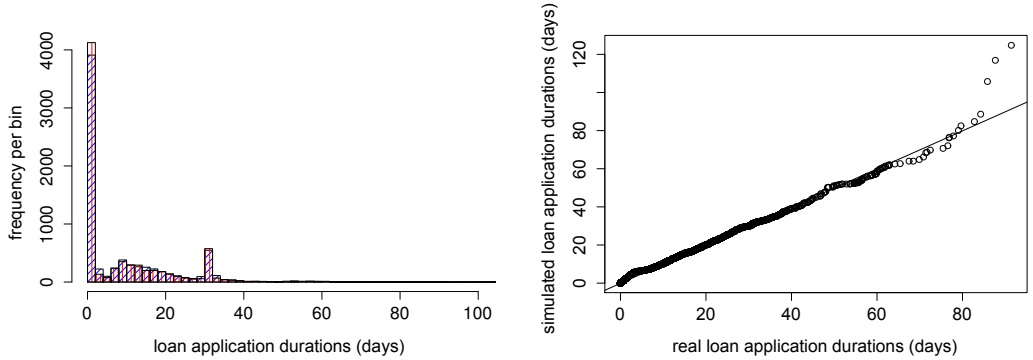
*Obtained Model Quality for the Shipment Import Process*

We use the same setting, as for the surgery model, i.e., the log-spline approximations to the timed transitions. In Figure 54, we compare the real durations from arrival of the container to pick up by inland transport with the simulated durations. Note that in this scenario we do not face the problem of missing entries, as in the surgery case. We also see that the acquired model is able to reproduce the observed behavior quite well.

The mean real duration is 4 days and 39 minutes. The mean simulated duration is 4 days and 8 minutes, which accounts for a bias of 0.54 percent. The standard deviation of the real duration is 54 hours and 14 minutes compared to 50 hours 51 minutes for the simulated traces. This accounts for a 6.26 percent lower standard deviation in the simulated cases. Although here, the model is not able to capture the full variance of the process, a two-sample Kolmogorov-Smirnov test for equality fails to reject the hypothesis that the two samples are from the same distribution with a p-value of 0.189. In other words, the probability that both the real traces and the simulated traces could have been generated by the same underlying distribution is not small enough to reject the hypothesis.

*Obtained Model Quality for the Loan Application Process*

Last, we obtain a GDT_SPN model for the loan application process and compare the simulated traces with the observed traces. Figure 55 shows the histograms in comparison and the quantile-quantile plot for the real traces and the simulated




(a) Comparison of histograms of observed real durations (red vertical stripes) and simulated durations (blue diagonal stripes).

(b) Quantile-quantile plot for the real loan application durations and the simulated durations.

Figure 55: Financial loan application process results. Except for a few outliers (simulated traces can take up to 120 days, whereas the longest real trace took about 80 days) the approximation of the learned model can be read from the overlap of the histograms or the rather straight line of the quantile-quantile plot on the right.

ones. The mean duration of the real traces is 8 days,18 hours and 53 minutes, compared to the mean simulated duration of 9 days, 1 hour and 41 minutes. This yields a bias of 3.2 percent for the mean. The standard deviation of the simulated durations is only 0.35 percent smaller than that of the real durations.

| | | real | simulated | bias |
|---|---|---|---|---|
| surgery process duration | mean | 3.364 h | 3.777 h | +12.28% |
| | sd | 2.840 h | 2.315 h | -18.49% |
| import process duration | mean | 4.027 d | 4.005 d | -0.55% |
| | sd | 2.261 d | 2.119 d | -6.28% |
| loan appl. process duration | mean | 8.787 d | 9.070 d | +3.22% |
| | sd | 12.642 d | 12.598 d | -0.35% |

Table 2: Means and standard deviations of durations. Units are either hours (h) or days (d). The bias is calculated as percentage value compared to the values of the real traces.

Table 2 summarizes the three use cases and shows the overview of the mean durations and sample standard deviations of the real case durations for each use case. These values are compared with the simulated traces sample statistics and the bias is shown as relative difference. We can read from the table that for the surgery case we get the worst approximation (i.e., the highest bias), but as we discussed earlier this is expected due to the high amount of missing entries. In fact, if we mine a process model from the traces, we get a spaghetti-like model with a lot of possible traces.

*Discussion*

In our sample of real-world processes, the independence assumption, i.e., the assumption that activity durations are independent, did not yield large biases in our models. We are unable to make a general claim to the validity of this assumption, because there might be cases where dependencies between activity durations are more prominent. Therefore, before using this framework, the process should be checked for strong dependencies between activity durations.

The worst results of the model were achieved for the surgery process. Here, we compare a log that faces the challenge of many missing events that are responsible for short case durations, with traces that fully comply with the model in figure 50 on page 122. Obviously, comparing full traces with parts of traces explains that in the real log, there are more traces that have very short durations, also increasing the overall variance.

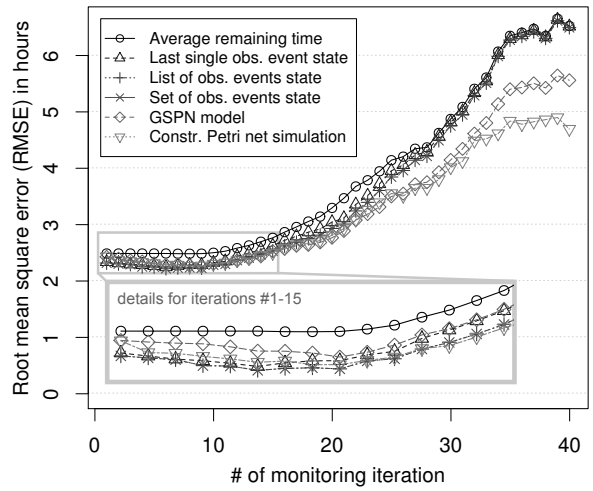
We argue that overall, the observed bias is acceptable given that in reality there can be many influential factors that cause side effects, e.g., dependencies between variables, or dependencies on other factors. In the extreme case that only the duration of one single activity is measured in a process, we can always produce a model with no bias of the mean, even if the distribution is bi-modal. To also capture the second moment (i.e., the variance) of the duration correctly, we use the logspline fitting approach [114]. This non-parametric approach can theoretically fit any distribution.

Our method has difficulties in capturing the variance in the import process. This indicates that the activity durations are positively correlated to some extent. Further investigation has shown that the correlation of the two durations recorded in the process (i.e., from arrival to discharge, and from discharge to pick-up) is 9.76 percent. Latter explains the reduced variance in the simulated traces, which is based on the assumption that the durations are independent.

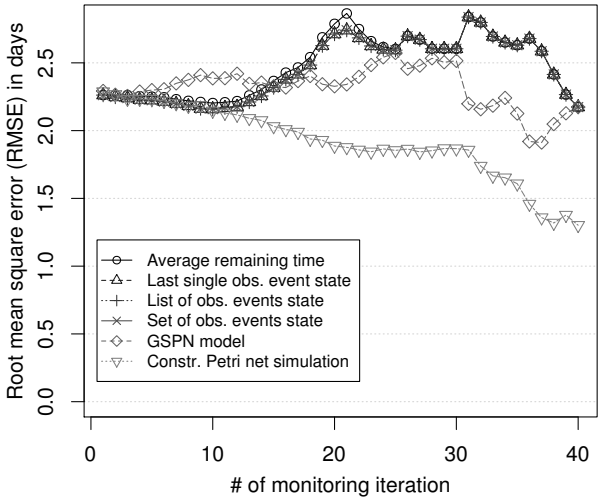
## 8.3 PREDICTED REMAINING DURATIONS

In this section, we evaluate the prediction approach presented in Chapter 4 with the real use cases. We follow the same experimental setup that we used for the conceptual evaluation of the prediction approach. The difference is that we do not have the theoretical model, and it is not guaranteed that we can capture the process characteristics with the obtained GDT_SPN model. Recall the experimental setup that is depicted in Figure 29 on page 74. Again, we perform a 10-fold cross validation using nine parts of the log as training data and one part as test data. We measure the prediction accuracy for each trace in at 2N points in time, such that the Nth point in time is the mean process duration. As in the conceptual evaluation of the prediction approach, we set N to 20.

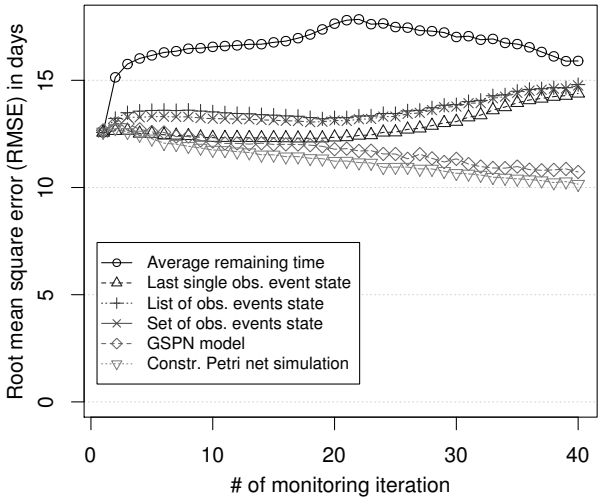
Figure 56 shows the results for the three industry case studies that we introduced in Section 8.1.




(a) Root mean square error (RMSE) in hours for the surgery process described at 40 periodic predictions.




(b) RMSE in days for the shipment import process at 40 periodic predictions.




(c) RMSE in days for the loan approval process [60] at 40 periodic predictions.

Figure 56: Prediction quality for logistics process. The root mean square errors are collected for periodic prediction iterations of the case studies. Thereby, a 10-fold cross validation is performed on the data with 40 periodic predictions, s.t. the 20th iteration is at the mean duration.

*Surgery Process Prediction Results*

Our first case study is the surgery process. As discussed in Section 8.1, we face a problem of missing data in the event log of the surgery process. To reduce the effect of missing data, we filter out non-fitting cases and use only the 570 cases that fit the model. Figure 56a shows the results for the surgery process. The first 15 iterations are zoomed in to show the details. It is remarkable that at the first iterations our prediction approach performs worse than the competing method based on annotated state transition systems [10]. Latter does not split the remaining process duration into individual independent activity durations as we do in our approach. Therefore, dependencies between activity durations are implicitly captured. At around the 13th iteration, by conditioning activity durations on the elapsed time, our approach outperforms the other approaches. Note that due to the small variability in the surgery process, the benchmark methods collapse to the same prediction quality. In comparison to the conceptual evaluation, the difference between the approaches is not that pronounced, however.

*Shipment Import Process Prediction Results*

Next, in Figure 56b, we evaluate the prediction approaches with the logistics process. This process only has three transitions, i.e., two time spans can be measured in between, cf. the model in Figure 51. Additionally, these two durations have a high variability. Our approach can capitalize on the additional knowledge of passed time, and the GDT_SPN model with conditional distributions yields significantly better results than the comparison methods.

For this use case, the different abstraction levels of the comparison method [10] collapse to the same results, cf. Figure 56b, because the process is sequential. The GSPN method yields on average comparable results to the benchmark. We can observe, however, that our approach produces more accurate predictions of the remaining process duration.

*Loan Approval Process Prediction Results*

For the loan approval process the prediction models—except the simplest model that only averages remaining time—produce comparable results until reaching the mean duration, i.e., the 20th monitoring iteration. For the cases that take longer than the average, the Petri net based prediction methods (i.e., the GSPN approach and our prediction approach using conditional probabilities in the GDT_SPN model) outperform the benchmark method based on annotated state transition systems [10].

Note that the simpler GSPN based method, that is, using a model with exponential distributions for the activities, performs almost as well as our more flexible approach. This indicates that the process activities are well approximated with the exponential distribution. The implicit assumption of the exponential distribution applies here, that is, events have a certain rate of occurrence, and that rate is independent of how long we waited before.

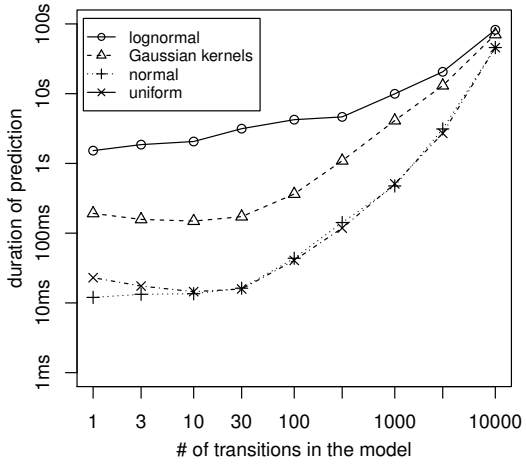



Figure 57: Prediction algorithm performance. Prediction durations for differently sized randomly generated, acyclic GDT_SPN models. Axes in log-scale.

Another characteristic of this process is that the best average prediction is not accurate, as the error is in a range of 10–15 days.

*Summary*

To summarize the qualitative evaluation results, the following characteristics of our approach can be observed:

- At early prediction periods, the approach performs about as well as the benchmarks.
- The improvements become more significant as time proceeds.
- The approach is most valuable for long running cases, which are critical to be detected and avoided.

*Scalability Analysis*

We propose to use simulation as means to predict the remaining service execution time for a running case. Therefore, it is interesting to see how long the prediction approach takes to simulate the remaining behavior of the process. To test for scalability, we conduct the following experiment.

First, we randomly generate structured, acyclic GDT_SPN models of different sizes by iterative insertion of sequential, parallel, and exclusive blocks, until we reach the desired node size of the net. For each timed transition we specify a distribution, i.e., either *uniform*, *normal*, or *lognormal* distributions, or non-parametric *Gaussian kernel* density estimators based on a random number of observations. In general, the run-time of the prediction algorithm depends on multiple factors:

- The *average number of transitions* that need to fire to reach the end of the process, influenced by the size of the net, the progress of the current case, and potential cycles.

- The *kind of transition distributions*, as there exist distributions that are rather costly to draw samples from, e.g., complicated non-parametric models, as well as very simple models, e.g., the uniform distribution.
- The *requested accuracy of the prediction*. Besides the fact that computing a narrow confidence interval takes more samples than allowing more sampling error, the variance of the process durations also influences the number of samples required to achieve the requested precision.
- The *computing power* of the system running the simulation.

For our experiments, we fixed the *requested accuracy* to a 99 percent confidence interval within maximum $\pm 3$ percent of error of the mean remaining duration. Regarding *computing power*, we used a regular laptop computer with a Pentium i7 620M (2.66 GHz cores) equipped with 8GB of ram. We varied the other two factors, i.e., the *average number of transitions* and the *kinds of distributions* used. Figure 57 depicts the average time taken for remaining time prediction of acyclic GDT_SPN models based on the number of transitions in the model in log-scale. For example, predicting the duration of a medium sized model (100 transitions) takes around 300 milliseconds for rather expensive non-parametric Gaussian kernel density estimation. Prediction of models with lognormally distributed values takes long because of higher variance. Note that a prediction of a model with 10000 transitions still is feasible in less than 100 seconds with these configurations.

In our experience, most business processes involving human activities take hours, days, or sometimes even months to complete. In these situations, the quality of the prediction is more important than the performance of the prediction approach. If performance is critical however, the approach could be extended to provide a fall back option to an analytical method based on GSPN models, as implemented in [218].

## 8.4 REPAIRED MISSING DATA

In this section, we examine how well the missing data imputation works in a real setting. We motivated the imputation approach that we presented in Chapter 5 with our observations in manual processes, i.e., we encountered around ten percent of forgotten documentation in a hospital event log [110]. Now, we use that real-life log and investigate how well we can restore *artificially forgotten* events.

The experimental setup is depicted in Figure 58. We do a similar experiment, as in the conceptual part (see Figure 35). The problem is that we do not know, if an event that is missing from the event log is a documentation error, or if the corresponding activity was not performed. Even with our assumption of a *normative model* (i.e., that the model captures the execution), we have no means to evaluate how well our imputed values fit the supposedly missing events. Therefore, we need to conduct a controlled experiment. To obtain actual values to compare our repaired results with, we first acquire traces that fit the model. We perform a 10-fold cross-validation on these traces—that is, we split them into a training log and a test log. We use the training log to learn the GDT_SPN model and the test log to randomly remove events and pass the log with missing data

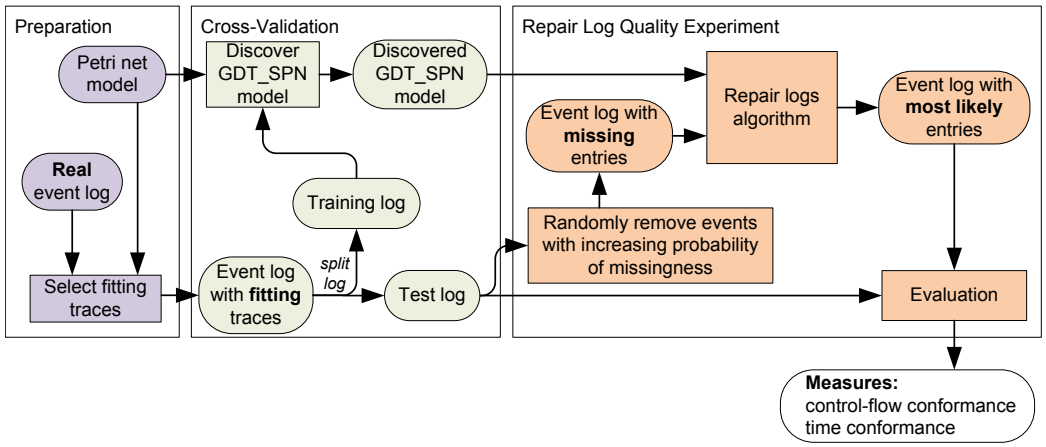



Figure 58: Setup to evaluate repair quality of case studies. We select the fitting traces from the real event log. With that log, we do a cross validation, and learn the performance model from the training log. From the test log, we randomly remove events (maintaining at least one event per trace), and repair them according to the model. We evaluate how well the repaired entries match the original entries in the trace.

to the repair algorithm with the learned GDT_SPN model. The remaining part is equivalent to the conceptual evaluation. This way, we are able to see how the repair algorithm performs in realistic settings.

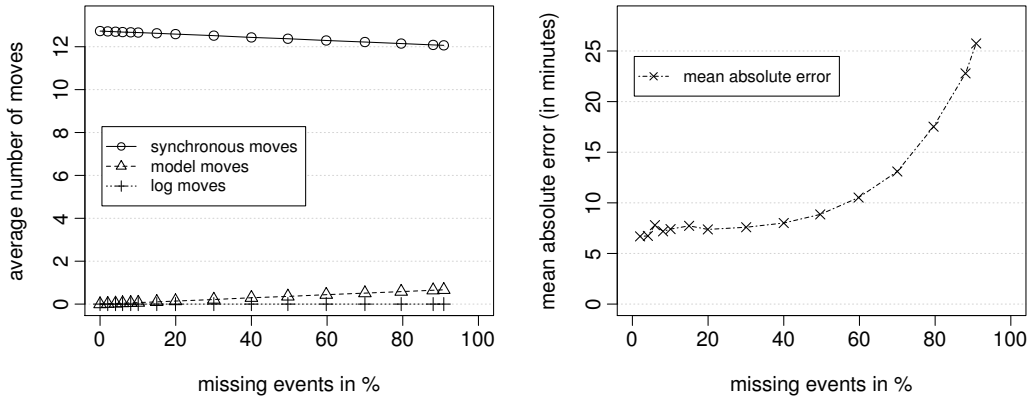
*Repairing the Surgery Process Event Log*

We first look at the results obtained from repairing the event log of our surgery case study.

The log of this treatment process contains missing entries, which motivated the work that we presented in Chapter 5. Out of 1310 patient treatment cases, only 570 fit the model shown in Figure 50 perfectly. The other cases contain one or more missing events. We use the 570 fitting cases to evaluate how well we can repair them after randomly removing events.

Figure 59 shows the evaluation results with the real log. Observe that the structure can be repaired quite well in comparison to the example used in the conceptual evaluation in Figure 38 on page 94. This is due to the sequential nature of the model with twelve events in sequence and two activities that are optional. This process model is rather restrictive, and shows a process that is standardized and linear in its execution. The number of synchronous moves gradually approaches twelve, as the amount of missing events increases. This is caused by the inability of the algorithm to repair single undetected events that are optional (e.g., the event *perform antibiotics prophylaxis* cannot be repaired, if it is missing).

The mean absolute error in the restored events is higher than the artificial example. This value greatly depends on the variance in the activity durations. In the evaluation example, the variance of certain activity durations in the model is quite high, e.g., the duration of the *start of emergence* activity has a mean of 4.69




(a) Number of synchronous, model, and log moves depending on the number of missing events from the log.

(b) The mean absolute error of the repaired times (in minutes).

Figure 59: Repair results for the surgery case study. The x-axis shows the amount of noise that is introduced into the event log. Each point on this axis represents an iteration of repair with a certain amount of introduced noise. We require at least one event in each trace, which means that we cannot increase the amount of missing events to 100 percent.

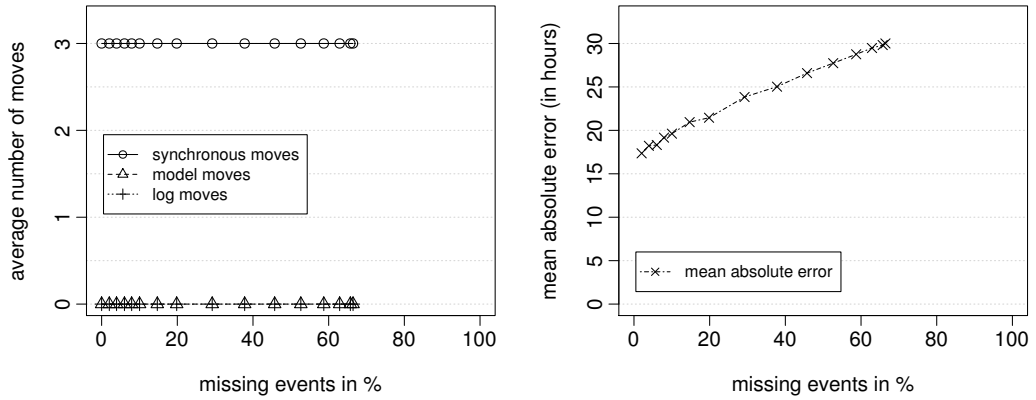
hours and variance of $18.29^2$. The data underlying this model is right-skewed with most values small, and some outliers. Here, the normal distribution is inappropriate, and other distributions might fit better.

Obviously, the ability to repair a log highly depends on the information content of observed events in the trace and the remaining variability in the model. For instance, we can always repair a pure sequential model with fitness 1.0 of the repaired log, even if we observe just one activity. However, the chance to pick the same path through a model composed of $n$ parallel activities with equally distributed times is only $\frac{1}{n!}$.

The deterministic repair mode, which we evaluated, is unable to restore optional branches without structural hints, i.e., at least one activity on that optional branch needs to be recorded. This affects single optional activities most, as their absence will not be repaired. However, many real-life processes are sequential, and can be repaired correctly.

*Repairing the Shipment Import Process*

Figure 60 shows the results of performing the experiment on the shipment import process depicted in Figure 51. The structural quality of the traces (Figure 60a) can be restored without error, which is obvious, as there is no other possibility than the prescribed order of the three events. The figure on the right, i.e., Figure 60b, is more interesting and shows that the error of the repaired times linearly grows with the amount of noise. This can be due to the nature of the distributions, as well as to the fact that the model only consists of three sequential transitions. Here, in the worst case only one event remains per trace, such




(a) Number of synchronous, model, and log moves depending on the number of missing events from the log.

(b) The mean absolute error of the repaired times (in hours).

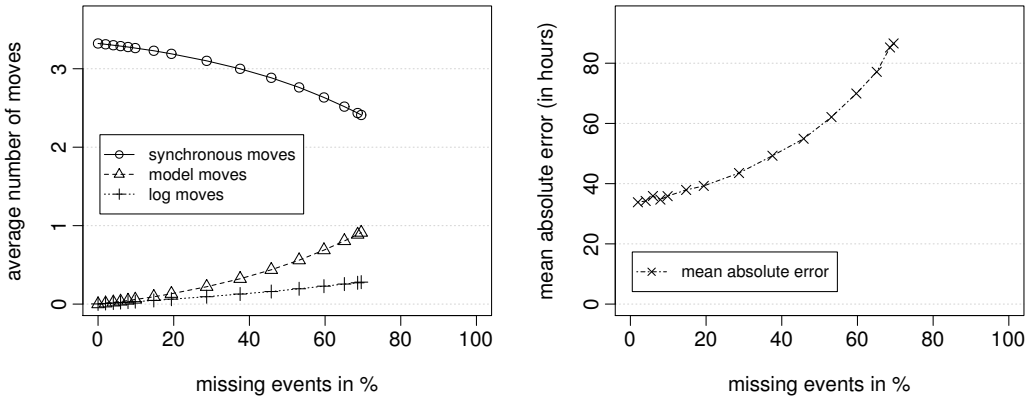
Figure 60: Repair results for the shipment import process. Depicted are the repaired quality of the structure (a) and the timestamps (b). The structure of a purely sequential model can be completely restored, even if only a single event remains in a trace. The timestamps of the missing events can be restored to an average error of under 20 hours with no noise. The quality decreases in a linear relationship to the amount of missing events.

that, missing events are either direct neighbors of the remaining event, or apart by one missing event.

*Repairing the Loan Application Process Event Log*

In Figure 61, we depict the results for repairing missing events in the loan application process. Here, the event logs are generated by a process execution engine, which ensures that the event logs fit the model. Nonetheless, it is interesting from a theoretical perspective to evaluate how well we could repair the traces if entries were amiss. We perform the same experimental setup as in the previous case studies, and in the conceptual evaluation of the repair method in Chapter 5. Figure 61a shows similar characteristics as the previous case studies. The ability to restore the correct events is decreasing with the number of missing events. Notice that we are unable to remove more than around 70 percent of the events, because we always keep at least one event of each trace in this experiment.

Figure 61b also shows the non-linear growth of the error of the time that was restored. Even at lower noise levels, the average error of repairing one missing event in a trace is around one and a half days. In regard of the high variance of the duration of this process—many cases finish in less than one day (due to early rejects), while some applications might take up to 80 days—this result is not surprisingly bad. Therefore, if the participants documented the process by hand and the *accept loan* event was missing from documentation, it would be helpful to be reminded, whether they performed the *accept loan* activity around a certain likely day.




(a) Number of synchronous, model, and log moves depending on the number of missing events from the log.

(b) The mean absolute error of the repaired times (in days).

Figure 61: Repair results for the loan application process. In the loan application process, the mean absolute error is non-linearly increasing with the amount of noise. The average number of moves is rather low (slightly above 3) for a process that has five transitions on the main path. This indicates that many cases are almost immediately declined or canceled.

*Discussion*

In our case studies, the repair algorithm exhibits similar characteristics. That is, it can tell us rather well *which* events are missing. The error in the question *when* these events have happened needs to be measured in relation to the variance of single activity durations. Given the assumptions that activity durations are independent, and without additional information (e.g., schedules, interdependencies between instances), the proposed method, that conditions on the observed events' occurrence times and computes the expected occurrence based on the posterior probability, is the best estimation possible.

We need to point out some limitations of this experiment, however. The missing data problem has been motivated by our observations in manual process execution environments, where documentation is done by hand. This is only the case in the surgery case study. The other two case studies are supported by IT systems and do not suffer from missing data issues. In the surgery case, we removed 770 inconsistent cases from the 1310 cases in the experiment. We are unable to judge the repair quality of the events missing in these 770 cases, because we do not know the actual time values of these events. It is also not possible to restore the true time information of the missing events, because the event log is from the year 2011.

Therefore, we perform a controlled experiment, where we take intact traces and simulated the missingness mechanism. We cannot make conclusions about the missingness mechanism causing the missing events from looking at a Petri net model and an event log, in which certain events that are required by the model are missing. However, we make the assumption that data is missing at random (MAR), or missing completely at random (MCAR), (see Section 5.1).

More investigation into the causes of missing data in process event logs is necessary, however.

We cannot exclude that—for some reason—the absence of an event from a case depends on the activity duration. Latter corresponds to the not missing at random (NMAR) case, in which special correction mechanisms are required [21].

Also, in this evaluation, we assumed that our model accurately represents the process behavior. Therefore, we only selected the part of the real event log that fits the model. We did this to eliminate another source of noise in the experiment, i.e., the error that is introduced by estimating the model parameters from noisy data.

First, we assume that the model contains the truth about the process, while in many cases there might not be such a model, e.g., in cases where the process is constantly evolving. We assume also that the timing information of the events in the event log are correct. That is, we do not test for plausibility of the existing events, but rather try to add missing events that are in accordance to the observations in the event log.

## 8.5 SELECTED MONITORING POINTS

When selecting event monitoring points (EMPs), we are interested in identifying those that are most helpful for predicting the remaining duration of the process. We can use this information to ease the burden of unnecessary manual documentation for process participants, but there is another more relevant motivation, when setting up a monitoring architecture in a distributed environment. In such a scenario, we assume that we could set up monitoring on all the activities, but to limit costs, we are satisfied with connecting a subset of the activities to the monitoring architecture. We presented an algorithm to select the optimal subset of monitoring points of a given size in Chapter 6.

### *Evaluating the Selection*

To evaluate the selection of monitoring points, we measure how much we can improve the prediction quality by selecting monitoring points *optimally* in contrast to selecting them *randomly*. We then measure the prediction performance of our prediction algorithm at equal intervals, as we also did in Section 8.3.

Figure 62 shows the prediction quality depending on the number of event monitoring points (EMPs) for the surgery process case study. Due to the large number of combinations that are possible, there are 462 combinations to select 6 out of 11 possible EMPs, we enumerate all possible values only if the number of combinations is less than 100, otherwise, we randomly select a hundred sample combinations. There is a clearly visible trend that shows that on average, the number of installed monitoring points improves the prediction quality. There is an interesting effect visible in this plot, that shows that it is important for prediction quality *where* we install EMPs in the process, respectively which activities we decide to document. It turns out in this case study, that if we do not select the optimal EMPs, the overall prediction accuracy can even *decrease* although we

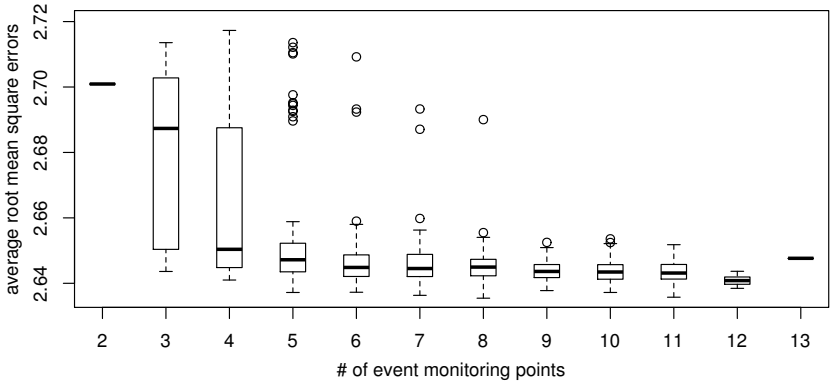



Figure 62: Prediction errors depending on number of EMPs. The graph shows box plots, where each sample is representing a configuration of monitoring points. For example, for 2 EMPs there is only one configuration, for 3 EMPs there are 11 combinations (start and end fixed, the remaining EMP can be at each transition in between). Each sample point captures the average prediction accuracy of 40 periodic predictions, i.e., the average of the root mean square errors of the predictions of the surgery process. We have removed the optional order patient event from this analysis, so that all cases start with the arrival in the lock.

increase the number of EMPs. This unexpected result is visible in Figure 62 if we compare the prediction results of 2 and 3 selected EMPs respectively.

*Discussion*

We investigated this matter, and it turned out that selecting only the *perform antibiotics prophylaxis* EMP, besides the EMPs at the start and end of the process, decreases the prediction accuracy. This is caused by the implicit assumption in obtaining the process performance characteristics, that activity durations are independent from the path that was taken. Thus, if we only measure the start (i.e., *arrival in lock*), the optional activity *perform antibiotics prophylaxis*, and the end of the process (i.e., *departure of recovery*), the time estimation of departure of recovery contains the durations from the start to end (collected from cases without antibiotics treatment), and from the optional activity (in cases antibiotics were administered). For a better model, the times need to be separated, i.e., for the cases that antibiotics prophylaxis is performed, the remaining time needs to be separately estimated, than if it is not performed. Another effect that can be responsible for this counterintuitive behavior is that by only measuring start and end of the process, all dependencies between activities are implicitly taken into account.

This evaluation shows the necessity to optimize the selection of EMPs, when installing a monitoring environment. The prediction quality with three optimally selected EMPs is about as good, as when installing all thirteen EMPs. This result shows that it is possible to get better prediction results by optimally selecting four EMPs, than by choosing eleven EMPs in the least effective combination.

# 9 CONCLUSION

Concluding this thesis, we provide a summary of the contributions of this thesis in Section 9.1. Last, we discuss limitations and future work in Section 9.2.

## 9.1 SUMMARY

With this thesis, we have introduced an advanced probabilistic model to business process management that is motivated by the need for probabilistic handling of unobserved process events in the healthcare domain. The probabilistic model is based on the Petri net formalism and allows us to capture arbitrary duration distributions of activities, and path probabilities. The model can be applied throughout the business process lifecycle. We support i) to *discover the performance model*, ii) to *predict the remaining duration* with it, iii) its potential to *ensure the documentation quality*, and iv) to *optimize the placement of monitoring points* in a business process. Let us briefly summarize each of these contributions in more detail.

*I) Discover the Performance Model*

First, we have highlighted different execution policies and model properties that pose challenges in discovery. We have reduced the problem caused by the race policy to the censored data problem in statistics. To estimate duration distributions with censored data, we have employed an existing implementation based on a non-parametric log-spline fitting technique [114].

The second challenge that we have addressed is to find the optimal assignment of weights to transitions in the GDT_SPN model that best explains the possibly contradictory observations of firing counts. We have approached the problem with a maximum likelihood estimation approach, which identifies the optimal weight assignments by optimizing a convex cost function with gradient descent.

*II) Predict the Remaining Duration*

Prediction of the time of a future process state—in our examples, we have been interested in the end event—of an already running case is a sparsely covered topic in business process management research. Most works focus on analysis of a whole process. In this thesis, we have used a probabilistic approach to achieve improved prediction results for single running instances. We have shown that conditional probabilities can be exploited in situations where we can condition on gained knowledge, e.g., when we know that a certain activity takes longer than a part of our historical observations.

*III) Ensure the Documentation Quality*

We have observed that in manual process execution environments, it is often the case that activities are executed in reality, but corresponding events are missing from documentation. A plausible cause for this is human error. Usually, in process mining the assumption is made that the event log contains the truth about the process, and there are no techniques addressing this mismatch.

We have presented a method that is built on a conformance checking technique, and have proposed a decomposition of the problem: to decide first *which* events are missing, and later to decide *when* those events happened most likely. Thereby, we have accepted the limitation that we follow a heuristic. The technique works as follows. The model is unfolded according to the chosen path and converted into a Bayesian network. In latter, we perform inference given the event occurrences to compute the most likely time values of the missing events.

*IV) Optimize the Placement of Monitoring Points*

Last, we have also faced the question, at which points in the process it makes most sense to install event monitoring points (EMPs). We have assumed that we already had some knowledge about the process and its activities. This knowledge can for example be provided by process participants. We have shown how we can capture the optimization in a Bellman equation that is suitable for a solution based on dynamic programming. We have investigated how much we can gain by optimal allocation of EMPs in the surgery use case. It has turned out in this case that we can achieve comparable prediction accuracy when optimally selecting three EMPs, as when selecting eleven EMPs in a non-optimal combination.

*Implementation and Evaluation*

We have implemented the contributions in ProM as plug-ins that are publicly available and have evaluated the approaches with real-world process data. Beside the health care domain that has served as motivation, we have checked the applicability of the methods to other domains, such as finance and logistics. The evaluation has indicated that the model is able to capture the different performance models in the encountered cases with low degree of bias. Also the prediction performs comparable to related work at early points in time of an instance, but can outperform the other approaches the more time passes.

## 9.2 LIMITATIONS AND FUTURE WORK

Rosenblueth and Wiener argue that "no substantial part of the universe is so simple that it can be grasped and controlled without abstraction." [180]. In fact, most of the limitations of this thesis are caused by the choice of abstraction. We have chosen to represent processes as a form of Petri nets. We have abstracted from dependencies between activity durations, from changes in the environment (i.e.,

*concept drift* in the data mining terminology), from resources and their working schedules, and from data in the processes. But even with all these abstractions, the performance of a process can be captured statistically in the models, as we have evaluated in Section 8.2. In the following, we discuss how we can reduce the level of abstraction to better capture reality.

*Petri Net Model*

A limitation of our approach is caused by the choice of representation. By choosing Petri nets, we disregard complex workflow patterns, such as the synchronizing merge [8] that is supported in the OR-join construct in BPMN [150]. This limitation is often not relevant, however, as empirical work by zur Mühlen and Recker [144] suggests that a large share of process models consists of only basic workflow constructs that can be expressed in Petri nets. Nonetheless, an extension of the probabilistic model to more complex workflow constructs seems promising. Following up on the Pfeffer's idea to use dynamic Bayesian networks for process modeling [160], a direct translation from BPMN models into Bayesian networks should be investigated in future work. There are unresolved research questions when using dynamic Bayesian networks, however. For example, how can we predict future states, when it is not clear yet, how often a loop in the process model will be traversed.

*Dependencies Between Activity Durations*

As we have observed in the event log of our logistics use case, the two durations of the time from arrival of a seavessel to discharge and from discharge to pick up by inland transport are correlated with a correlation coefficient of 0.0976 (i.e., 9.76 percent). See the discussion in Section 8.2. Such dependencies introduce a bias in our method, because the GDT_SPN model is unable to represent such dependencies. One way to extend the model is to make each activity aware of the history, as in the proposal by Schonenberg et al. in [191]. That is, to condition probabilistic decisions on previous decisions in the case. For example, this allows us to express that the probability to leave the loop rises with each iteration. The work in [191] is not dealing with duration aspects, however, but focuses on choices in the routing of a case in the model. The problem that makes it difficult to automatically derive such a more flexible model is that one needs to decide, on *which* events in the past to condition the probabilities and the probability distributions.

If we assume that *all* events in the history can influence the current activity duration, we are facing the curse of dimensionality, see also Bishop's discussion on the curse of dimensionality for machine learning problems in [38, Section 1.4]. Each single variable increases the combinations by another dimension. The number of samples that we need to estimate such a multivariate distribution correctly grows exponentially with each additional variable (i.e., in our case with each additional activity to condition on). Techniques to reduce the dimensions, or to

abstract from them, are required, and future work should address the problem of finding good abstraction criteria.

*Resources and Schedules*

In this thesis, resources (e.g., actors) and schedules are out of scope. Resources and their schedules play an important role in business process management, however, and we expect substantial improvements of model accuracy by explicitly capturing resources. The proposed method is based on Petri nets, which offers the capability to add resources to the model in a natural way. One possibility is to model resources as tokens. With the current firing semantic of GDT_SPN models (i.e., firing of timed transitions is instantaneous and atomic), each activity that is performed by a resource needs at least two transitions. That is, one transition starting the work (to block the required resources), and a second one ending the work (to release the blocked resources). Thus, we need to model waiting times and execution times of activities separately, and we need to know the resource constraints of the activities and the available resources of the process execution environment. If this additional information is available, resource integration is possible without the need to extend the GDT_SPN model.

If we wanted to also integrate resource *schedules* into the GDT_SPN model, we would require an extension of the formalism, as the execution semantics that we have considered do not support a global clock, and constraints. An option could be to use the more expressive Coloured Petri net approach, as proposed by Jensen [98]. An open question is how to learn these models from event logs, but already there exists work on mining data aware model for simulation by Rozinat et al. [183]. Another question is how to integrate work breaks of process workers into the model. First steps to extract such interruptions from the data have been proposed in [212].

Enriching the model with less detailed workload information (i.e., by categorizing cases based on current workload in the system), can reduce prediction errors significantly, as Folino et al. have shown [70]. We expect to achieve comparable improvements by applying the former refinement to our model. A comparison between this coarse grained system load model with a full resource model with queuing would be very interesting in order to see if it is reasonable to spend more effort on making the model more detailed.

*Concept Drift Detection*

We have mentioned in our assumptions that we assume the business process to be in a steady state. A successful application of the model in real-world settings requires to dismiss this assumption, because real-world business processes are subject to many different external influences. For example, changes in average arrival rates of patients range from two patients per hour in the night to twelve patients per hour around noon on Sundays [126]. For such recurring changes, further research might explore a similar approach as in [70], that is, to cluster

the cases based system load and obtain multiple models that are then used in the respective system state.

For gradual changes (e.g., evolution of the process, or of the market) we envision a system that has a concept drift detection component added to the architecture that makes sure that concept drifts are detected. When detecting that the current model is no longer accurate, a new calibration of the GDT_SPN model can be triggered to adapt it to the changed environment.

An example method is provided by Gama et al. in [71] that indicates a drift if a threshold for wrong classifications is breached, and relearns the probabilistic model from the cases after that breach to improve successive estimations. An application of this idea to the continuous time domain used in this thesis could be the following: we categorize the cases as fitting to the GDT_SPN model, if the probability of their occurrence is above a certain threshold according to the model, and as not fitting otherwise. Then, if we encounter *many* cases that are not fitting to the model (i.e., a number above a threshold), we can update the model.

Further methods to not react to, but anticipate concept drifts also exist, e.g., as Yang et al. propose in [213]. Latter work keeps a collection of contexts in a history and tries to anticipate the next changes based on patterns observed in the past.

*Most Likely Alignment between Log and* GDT_SPN *Model*

The cost-based alignment method is time-agnostic [15]. We have made but a first step towards finding the most likely alignments by considering the path probabilities in the selection of the alignment. The problem, however, is not solved yet entirely. As we have mentioned in the discussion of the problem complexity in Section 5.2, the inclusion of time in the search for the most likely alignments is increasing the number of solutions to the uncountable infinite domain. Moreover, each time value of missing events affects the joint likelihood of the events in the network. This poses significant challenges to a deterministic search.

Nevertheless, we are convinced that it is possible to solve this intriguing and challenging problem, and we will tackle it in future work.

# BIBLIOGRAPHY


[1] Wil M. P. van der Aalst. Verification of Workflow Nets. In *ICATPN'97*, volume 1248 of *LNCS*, pages 407–426. Springer, 1997.

[2] 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.

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

[4] Wil M. P. van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos Buijs, et al. Process Mining Manifesto. In *Business Process Management Workshops*, volume 99 of *LNBIP*, pages 169–194. Springer, 2012.

[5] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying History on Process Models for Conformance Checking and Performance Analysis. In *WIREs: Data Mining and Knowledge Discovery*, volume 2, pages 182–192. Wiley Online Library, 2012.

[6] Wil M. P. van der Aalst, Kees M. van Hee, and Hajo A. Reijers. Analysis of Discrete-time Stochastic Petri Nets. *Statistica Neerlandica*, 54(2):237–255, 2000.

[7] Wil M. P. van der Aalst and Arthur H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information systems*, 30(4):245–275, 2005.

[8] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow Patterns. *Distributed and parallel databases*, 14(1):5–51, 2003.

[9] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske. Business Process Management: A Survey. *Process Management*, 42(2):119–128, May 2003.

[10] Wil M. P. van der Aalst, M. Helen Schonenberg, and Minseok Song. Time Prediction Based on Process Mining. *Information Systems*, 36(2):450–475, 2011.

[11] Wil M. P. van der Aalst, Boudewijn F. van Dongen, Joachim Herbst, Laura Maruster, Guido Schimm, and Anton J. M. M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data & knowledge engineering*, 47(2):237–267, 2003.

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

[13] Andre Acusta, Paul Eggermont, and Vincent Lariccia. An EM Algorithm for Density Estimation with Randomly Censored Data. *Journal of Statistical Computation and Simulation*, 73(3):223–232, 2003.

[14] Arya Adriansyah and Joos C. A. M. Buijs. Mining Process Performance from Event Logs – The BPI Challenge 2012 Case Study. sumitted to BPI Challenge 2012 `http://www.win.tue.nl/bpi2012/lib/exe/fetch.php?media=adriansyah.pdf`, 2012.

[15] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Conformance Checking Using Cost-Based Fitness Analysis. In *EDOC 2011*, pages 55–64. IEEE, 2011.

[16] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Memory-Efficient Alignment of Observed and Modeled Behavior. Technical Report BPM-13-03, BPM Center Report, 2013.

[17] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining Process Models from Workflow Logs. In *Advances in Database Technology — EDBT'98*, volume 1377 of *Lecture Notes in Computer Science*, pages 467–483. Springer Berlin / Heidelberg, 1998.

[18] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining Association Rules Between Sets of Items in Large Databases. *SIGMOD Record*, 22(2):207–216, June 1993.

[19] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-Based Learning Algorithms. *Machine learning*, 6(1):37–66, 1991.

[20] Hirotsugu Akaike. A New Look at the Statistical Model Identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723, 1974.

[21] Paul D. Allison. *Missing Data*. Number 136 in Quantitative Applications in the Social Sciences. Sage, Thousand Oaks, CA, 2001.

[22] Martin Alt, Andreas Hoheisel, Hans-Werner Pohl, and Sergei Gorlatch. A Grid Workflow Language Using High-Level Petri Nets. In *Parallel Processing and Applied Mathematics*, pages 715–722. Springer, 2006.

[23] Rainer von Ammon, Christoph Emmersberger, Torsten Greiner, Adrian Paschke, Florian Springer, and Christian Wolff. Event-Driven Business Process Management. In *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*, 2008.

[24] Nikolas Anastasiou, Tzu-Ching Horng, and William J. Knottenbelt. Deriving Generalised Stochastic Petri Net Performance Models from High-Precision Location Tracking Data. In *VALUETOOLS'11*, pages 91–100. ICST, 2011.

[25] Kaiomars P. Anklesaria and Zvi Drezner. A Multivariate Approach to Estimating the Completion Time for PERT Networks. *The Journal of the Operational Research Society*, 37(8):811–815, 1986.

[26] Ahmed Awad, Matthias Weidlich, and Mathias Weske. Visually Specifying Compliance Rules and Explaining their Violations for Business Processes. *Journal of Visual Languages & Computing*, 22(1):30–55, 2011.

[27] Bürgerliches Gesetzbuch (BGB). § 630f Dokumentation der Behandlung, 2013.

[28] Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer. A Maximum Likelihood Approach to Continuous Speech Recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Pami-5(2):179–190, March 1983.

[29] Thomas Baier and Jan Mendling. Bridging Abstraction Layers in Process Mining by Automated Matching of Events and Activities. In Florian Daniel, Jianmin Wang, and Barbara Weber, editors, *Business Process Management*, volume 8094 of *Lecture Notes in Computer Science*, pages 17–32. Springer Berlin Heidelberg, 2013.

[30] Gianfranco Balbo. Introduction to Stochastic Petri Nets. In *Lectures on Formal Methods and Performance Analysis*, pages 84–155. Springer, 2001.

[31] Gianfranco Balbo and Giovanni Chiola. Stochastic Petri Net Simulation. In *Proceedings of the 21st conference on Winter simulation*, pages 266–276. ACM, 1989.

[32] David W. Bates, Michael Cohen, Lucian L. Leape, J. Marc Overhage, M. Michael Shabot, and Thomas Sheridan. Reducing the Frequency of Errors in Medicine Using Information Technology. *Journal of the American Medical Informatics Association*, 8(4):299–308, 2001.

[33] Eric Becker, Vangelis Metsis, Roman Arora, Jyothi Vinjumur, Yurong Xu, and Fillia Makedon. SmartDrawer: RFID-Based Smart Medicine Drawer for Assistive Environments. In *Proceedings of the 2nd International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '09, pages 49:1–49:8, New York, NY, USA, 2009. ACM.

[34] Piergiorgio Bertoli, Mauro Dragoni, Chiara Ghidini, and Di Francescomarino, Chiara. Reasoning-based Techniques for Dealing with Incomplete Business Process Execution Traces. Technical report, Fundazione Bruno Kessler, Data & Knowledge Management, 2013. `https://dkm.fbk.eu/images/9/96/TR-FBK-DKM-2013-1.pdf`.

[35] Henry H. Bi and J. Leon Zhao. Applying Propositional Logic to Workflow Verification. *Information Technology and Management*, 5(3-4):293–318, 2004.

[36] Patrick Billingsley. *Probability and Measure*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2012.

[37] Jonathan Billington, Søren Christensen, Kees Van Hee, Ekkart Kindler, Olaf Kummer, Laure Petrucci, Reinier Post, Christian Stehno, and Michael Weber. The Petri Net Markup Language: Concepts, Technology, and Tools. In *Applications and Theory of Petri Nets 2003*, pages 483–505. Springer, 2003.

[38] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[39] Andrea Bobbio and Miklós Telek. Computational Restrictions for SPN with Generally Distributed Transition Times. In *Dependable Computing—EDCC-1*, volume 852 of *LNCS*, pages 131–148. Springer, 1994.

[40] Diana Borrego, María T. Gómez-López, Rafael M. Gasca, and Rafael Ceballos. Determination of an Optimal Test Points Allocation for Business Process Analysis. In *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, pages 159–160. IEEE, 2010.

[41] Diana Borrego, María Teresa Gómez-López, and Rafael M. Gasca. Minimizing Test-Point Allocation to Improve Diagnosability in Business Process Models. *Journal of Systems and Software*, page (to appear), 2013.

[42] George E. Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, 2 edition, 1976.

[43] Troyen A. Brennan, Lucian L. Leape, Nan M. Laird, Liesi Hebert, A. Russell Localio, Ann G. Lawthers, Joseph P. Newhouse, Paul C. Weiler, and Howard H. Hiatt. Incidence of Adverse Events and Negligence in Hospitalized Patients: Results of the Harvard Medical Practice Study I. *New England journal of medicine*, 324(6):370–376, 1991.

[44] Lawrence Brown, Noah Gans, Avishai Mandelbaum, Anat Sakov, Haipeng Shen, Sergey Zeltyn, and Linda Zhao. Statistical Analysis of a Telephone Call Center: A Queueing-Science Perspective. *Journal of the American Statistical Association*, 100(469):36–50, 2005.

[45] Robert Buchholz, Claudia Krull, and Graham Horton. Reconstructing Model Parameters in Partially-Observable Discrete Stochastic Systems. In *Analytical and Stochastic Modeling Techniques and Applications*, pages 159–174. Springer, 2011.

[46] Joos C.A.M. Buijs, Marcello. La Rosa, H.A. Reijers, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Improving Business Process Models using Observed Behavior. In *Post-Proceedings of SIMPDA 2012*, LNBIP. Springer, 2013. (to appear).

[47] Andrea Cangialosi, Joseph E. Monaly, and Samuel C. Yang. Leveraging RFID in Hospitals: Patient Life Cycle and Mobility Perspectives. *Communications Magazine, IEEE*, 45(9):18–23, 2007.

[48] Rafael Ceballos, Victor Cejudo, Rafael M. Gasca, and Carmelo Del Valle. A Topological-based Method for Allocating Sensors by Using CSP Techniques. In *Current Topics in Artificial Intelligence*, volume 4177 of *Lecture Notes in Computer Science*, pages 62–68. Springer, 2006.

[49] Stanley F. Chen and Joshua Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996.

[50] Hoon Choi, Vidyadhar G. Kulkarni, and Kishor S. Trivedi. Markov Regenerative Stochastic Petri Nets. *Performance Evaluation*, 20(1):337–357, 1994.

[51] Gianfranco Ciardo, Reinhard German, and Christoph Lindemann. A Characterization of the Stochastic Process Underlying a Stochastic Petri Net. *IEEE Transactions on Software Engineering*, 20(7):506–515, 1994.

[52] Gregory F. Cooper. The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks. *Artificial intelligence*, 42(2):393–405, 1990.

[53] Claire Costello and Owen Molloy. Towards a Semantic Framework for Business Activity Monitoring and Management. In *AAAI Spring Symposium: AI Meets Business Rules and Process Management*, pages 17–27, 2008.

[54] Adnan Darwiche. Inference in Bayesian Networks: A Historical Perspective. In Rina Dechter, Hector Geffner, and Joseph Y. Halpern, editors, *Heuristics, Probability and Causality – A Tribute to Judea Pearl*, volume 11. College Publications, 2010.

[55] Adnan Darwiche and Judea Pearl. On the Logic of Iterated Belief Revision. *Artificial Intelligence*, 89(1–2):1 – 29, 1997.

[56] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining Stream Statistics over Sliding Windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[57] Joseph M. DeFee and Paul Harmon. Business Activity Monitoring and Simulation. Technical report, BP Trends Newsletter, White Paper and Technical Briefs, 2004.

[58] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B*, 39(1):1–38, 1977.

[59] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and Analysis of Business Process Models in BPMN. *Information and Software Technology*, 50(12):1281–1294, 2008.

[60] Boudewijn F. van Dongen. BPI Challenge 2012 Dataset, 2012. `http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`.

[61] Boudewijn F. van Dongen, R. A. Crooy, and Wil M. P. van der Aalst. Cycle Time Prediction: When Will This Case Finally Be Finished? In *On the Move to Meaningful Internet Systems: OTM 2008*, pages 319–336. Springer, 2008.

[62] Boudewijn F. van Dongen, Ana Karla A. de Medeiros, H. M. W. (Eric) Verbeek, Anton J.M.M. Weijters, and Wil M. P. van der Aalst. The ProM Framework: A New Era in Process Mining Tool Support. In *Applications and Theory of Petri Nets 2005*, pages 1105–1116. Springer, 2005.

[63] Marek J. Druzdzel and Max Henrion. Effcient Reasoning in Qualitative Probabilistic Networks. In *Proceedings of the 11th National Conference on Artificial Intelligence, Washington*, pages 548–553, 1993.

[64] Joanne B. Dugan, Kishor S. Trivedi, Robert Geist, and Victor F. Nicola. Extended Stochastic Petri Nets: Applications and Analysis. In *Proceedings of the Tenth International Symposium on Computer Performance Modelling, Measurement and Evaluation*, pages 507–519. North-Holland Publishing Co., 1984.

[65] Johann Eder and Horst Pichler. Duration Histograms for Workflow Systems. In *Engineering Information Systems in the Internet Context*, pages 239–253, 2002.

[66] Dirk Fahland and Wil M. P. van der Aalst. Repairing Process Models to Reflect Reality. In *BPM*, volume 7481 of *LNCS*, pages 229–245. Springer, 2012.

[67] Massimo de Falco and Roberto Macchiaroli. Timing of Control Activities in Project Planning. *International Journal of Project Management*, 16(1):51–58, 1998.

[68] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. John Wiley & Sons, 3rd edition, 1968.

[69] Shahina Ferdous. *Multi Person Tracking and Querying with Heterogeneous Sensors*. PhD thesis, University of Texas at Airlington, 2012.

[70] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Discovering Context-Aware Models for Predicting Business Process Performances. In *On the Move to Meaningful Internet Systems: OTM 2012*, pages 287–304. Springer, 2012.

[71] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with Drift Detection. In *Advances in Artificial Intelligence–SBIA 2004*, pages 286–295. Springer, 2004.

[72] Everette S. Gardner. Exponential Smoothing: The State of the Art. *Journal of forecasting*, 4(1):1–28, 1985.

[73] Dan Geiger, Thomas Verma, and Judea Pearl. Identifying Independence in Bayesian Networks. *Networks*, 20(5):507–534, 1990.

[74] Reinhard German. Non-Markovian Analysis. In Ed Brinksma, Holger Hermanns, and Joost-Pieter Katoen, editors, *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*, pages 156–182. Springer Berlin Heidelberg, 2001.

[75] Reinhard German, Christian Kelling, Armin Zimmermann, and Günter Hommel. TimeNET: A Toolkit for Evaluating Non-Markovian Stochastic Petri Nets. *Performance Evaluation*, 24(1):69–87, 1995.

[76] George M. Giaglis, Ioannis Minis, Antonios Tatarakis, and Vasileios Zeimpekis. Minimizing Logistics Risk through Real-Time Vehicle Routing and Mobile Technologies: Research to Date and Future Trends. *International Journal of Physical Distribution & Logistics Management*, 34(9):749–764, 2004.

[77] E. Mark Gold. Complexity of Automaton Identification from Given Data. *Information and Control*, 37(3):302 – 320, 1978.

[78] Herman Heine Goldstine and John von Neumann. *Planning and Coding of Problems for an Electronic Computing Instrument*. Institute for Advanced Study, Princeton, New Jersey, 1948.

[79] Jan G. de Gooijer and Rob J. Hyndman. 25 Years of Time Series Forecasting. *International Journal of Forecasting*, 22(3):443–473, 2006.

[80] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*. Addison-Wesley Professional, third edition, 2005.

[81] Richard Grol and Jeremy Grimshaw. From Best Evidence to Best Practice: Effective Implementation of Change in Patients' Care. *The Lancet*, 362(9391):1225–1230, 2003.

[82] Brian Gugerty, Michael J. Maranda, Mary Beachley, V. B. Navarro, Susan Newbold, Wahnita Hawk, Judy Karp, Maria Koszalka, Steven Morrison, Stephanie S. Poe, and Donna Wilhelm. Challenges and Opportunities in Documentation of the Nursing Care of Patients. Documentation Work Group, Maryland Nursing Workforce Commission. Baltimore., May 2007. (accessible online at: `http://www.mbon.org/commission2/documentation_challenges.pdf`).

[83] Christian W. Günther and Wil M. P. van der Aalst. Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In *BPM*, volume 4714 of *LNCS*, pages 328–343. Springer, 2007.

[84] Michael Hammer and James Champy. Reengineering the Corporation: A Manifesto for Business Revolution. *Business Horizons*, 36(5):90–91, 1993.

[85] David J. Hand. Classifier Technology and the Illusion of Progress. *Statistical Science*, 21(1):1–14, 2006.

[86] Philip Heidelberger and Stephen S. Lavenberg. Computer Performance Evaluation Methodology. *Computers, IEEE Transactions on*, 100(12):1195–1220, 1984.

[87] Joachim Herbst and Dimitris Karagiannis. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. *Intelligent Systems in Accounting, Finance and Management*, 9(2):67–92, 2000.

[88] Nico Herzberg, Matthias Kunze, and Andreas Rogge-Solti. Towards Process Evaluation in non-Automated Process Execution Environments. In *Proceedings of the 4th Central-European Workshop on Services and their Composition, ZEUS*, pages 96–102, 2012.

[89] Nico Herzberg, Andreas Meyer, and Mathias Weske. An Event Processing Platform for Business Process Management. In *Enterprise Distributed Object Computing Conference, EDOC 2013*, Vancouver, Canada, (to appear). IEEE.

[90] András Horváth, Antonio Puliafito, Marco Scarpa, and Miklós Telek. Analysis and Evaluation of Non-Markovian Stochastic Petri Nets. In *Computer Performance Evaluation. Modelling Techniques and Tools*, pages 171–187. Springer, 2000.

[91] Haiyang Hu, Jianen Xie, and Hua Hu. A Novel Approach for Mining Stochastic Process Model from Workflow Logs. *Journal of Computational Information Systems*, 7(9):3113–3126, 2011.

[92] San-Yih Hwang, Haojun Wang, Jian Tang, and Jaideep Srivastava. A Probabilistic Approach to Modeling and Estimating the QoS of Web-Services-Based Workflows. *Information Sciences*, 177(23):5484–5503, 2007.

[93] R.J. Hyndman and A.B. Koehler. Another Look at Measures of Forecast Accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

[94] Marta Indulska, Peter Green, Jan Recker, and Michael Rosemann. Business Process Modeling: Perceived Benefits. In *Conceptual Modeling-ER 2009*, volume 5829 of *Lecture Notes in Computer Science*, pages 458–471. Springer Berlin Heidelberg, 2009.

[95] Petr Janeček, Jiři Mošna, and Pavel Prautsch. The Use of Coxian Distribution in Closed-Form Treatment of Stochastic Petri Nets. In *Proceedings of the 15th Mediterranean Conference on Control & Automation*, pages 1–4. IEEE, 2007.

[96] Christian Janiesch, Martin Matzner, and Oliver Müller. A Blueprint for Event-Driven Business Activity Management. In Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 17–28. Springer Berlin Heidelberg, 2011.

[97] Monique Jansen-Vullers and Mariska Netjes. Business Process Simulation–a Tool Survey. In *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark*, 2006.

[98] Kurt Jensen. Coloured Petri Nets and the Invariant-Method. *Theoretical computer science*, 14(3):317–336, 1981.

[99] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 1. Springer, 1996.

[100] Wei Jiang, Tom Au, and Kwok-Leung Tsui. A Statistical Process Control Approach to Business Activity Monitoring. *Iie Transactions*, 39(3):235–249, 2007.

[101] Diane Jordan, John Evdemon, et al. Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`, April 2007. OASIS Standard.

[102] Michael I. Jordan. Graphical Models. *Statistical Science*, 19(1):140–155, 2004.

[103] Bokyoung Kang, Dongsoo Kim, and Suk-Ho Kang. Periodic Performance Prediction for Real-time Business Process Monitoring. *Industrial Management & Data Systems*, 112(1):4–23, 2012.

[104] Edward L Kaplan and Paul Meier. Nonparametric Estimation from Incomplete Observations. *Journal of the American statistical association*, 53(282):457–481, 1958.

[105] Despina C. Kartson, Gianfranco Balbo, Susanna Donatelli, Giuliana Franceschinis, and Giuseppe Conte. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, Inc., 1994.

[106] Joost-Pieter Katoen. GSPNs revisited: Simple Semantics and New Analysis Algorithms. In *Application of Concurrency to System Design (ACSD), 2012 12th International Conference on*, pages 6–11. IEEE, 2012.

[107] Slava M. Katz. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35(3):400–401, 1987.

[108] Gerhard Keller, August-Wilhelm Scheer, and Markus Nüttgens. *Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK)*. Inst. für Wirtschaftsinformatik, 1992.

[109] Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Christoph J. Bussler. On Structured Workflow Modelling. In Benkt Wangler and Lars Bergman, editors, *Advanced Information Systems Engineering*, volume 1789 of *Lecture Notes in Computer Science*, pages 431–445. Springer Berlin Heidelberg, 2000.

[110] Kathrin Kirchner, Nico Herzberg, Andreas Rogge-Solti, and Mathias Weske. Embedding Conformance Checking in a Process Intelligence System in Hospital Environments. In Richard Lenz, Silvia Miksch, Mor Peleg, Manfred Reichert, David Riaño, and Annette Teije, editors, *Process Support and Knowledge Representation in Health Care*, volume 7738 of *Lecture Notes in Computer Science*, pages 126–139. Springer Berlin Heidelberg, 2013.

[111] Andreas Knöpfel, Bernhard Gröne, and Peter Tabeling. *Fundamental Modeling Concepts*. Wiley, West Sussex UK, 2005.

[112] Jana Koehler, Giuliano Tirenni, and Santhosh Kumaran. From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods. In *Sixth International Enterprise Distributed Object Computing Conference (EDOC'02)*, pages 96–106. IEEE, 2002.

[113] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.

[114] Charles Kooperberg and Charles J. Stone. Logspline density estimation for censored data. *Journal of Computational and Graphical Statistics*, 1(4):301–328, 1992.

[115] Edouard Leclercq, Dimitri Lefebvre, and Souleiman Ould El Mehdi. Identification of Timed Stochastic Petri Net Models with Normal Distributions of Firing Periods. In *Information Control Problems in Manufacturing*, volume 13, pages 948–953, 2009.

[116] Philipp Leitner, Branimir Wetzstein, Florian Rosenberg, Anton Michlmayr, Schahram Dustdar, and Frank Leymann. Runtime Prediction of Service Level Agreement Violations for Composite Services. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, pages 176–186. Springer, 2010.

[117] Richard Lenz and Manfred Reichert. IT Support for Healthcare Processes – Premises, Challenges, Perspectives. *Data & Knowledge Engineering*, 61(1):39–58, April 2007.

[118] Christoph Lindemann and Gerald S. Shedler. Numerical Analysis of Deterministic and Stochastic Petri Nets with Concurrent Deterministic Transitions. *Performance Evaluation*, 27–28:565–582, 1996.

[119] Roderick J. A. Little and Donald B. Rubin. *Statistical Analysis with Missing Data*. Wiley, 2nd edition, 2002.

[120] Niels Lohmann, H. M. W. (Eric) Verbeek, and Remco Dijkman. Petri Net Transformations for Business Processes - A Survey. In *Transactions on Petri Nets and Other Models of Concurrency II*, volume 5460 of *LNCS*, pages 46–63. Springer Berlin Heidelberg, 2009.

[121] Ruopeng Lu and Shazia Sadiq. A Survey of Comparative Business Process Modeling Approaches. In *Business Information Systems*, pages 82–94. Springer, 2007.

[122] David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2002.

[123] Linh Thao Ly, Stefanie Rinderle, and Peter Dadam. Integration and Verification of Semantic Constraints in Adaptive Process Management Systems. *Data & Knowledge Engineering*, 64(1):3–23, 2008.

[124] Alex Macario, Terry S. Vitez, Brian Dunn, and Tom McDonald. Where are the costs in perioperative care?: Analysis of hospital costs and charges for inpatient surgical care. *Anesthesiology*, 83(6):1138, 1995.

[125] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*, volume 999. MIT Press, Cambridge, Massachusetts, 1999.

[126] Yariv N. Marmor, Segev Wasserkrug, Sergey Zeltyn, Yossi Mesika, Ohad Greenshpan, Boaz Carmeli, Avraham Shtub, and Avishai Mandelbaum. Toward Simulation-Based Real-Time Decision-Support Systems for Emergency Departments. In *Simulation Conference (WSC), Proceedings of the 2009 Winter*, pages 2042–2053. IEEE, 2009.

[127] Marco Ajmone Marsan. Stochastic Petri Nets: An Elementary Introduction. In *Advances in Petri Nets 1989*, pages 1–29. Springer, 1990.

[128] Marco Ajmone Marsan, Gianfranco Balbo, Andrea Bobbio, Giovanni Chiola, Gianni Conte, and Aldo Cumani. The Effect of Execution Policies on the Semantics and Analysis of Stochastic Petri Nets. *IEEE Transactions on Software Engineering*, 15:832–846, 1989.

[129] Marco Ajmone Marsan and Giovanni Chiola. On Petri Nets with Deterministic and Exponentially Distributed Firing Times. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1987*, volume 266 of *Lecture Notes in Computer Science*, pages 132–145. Springer Berlin Heidelberg, 1987.

[130] Marco Ajmone Marsan, Gianni Conte, and Gianfranco Balbo. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM TOCS*, 2(2):93–122, 1984.

[131] Laura Măruşter and Nick R. T. P. van Beest. Redesigning Business Processes: A Methodology Based on Simulation and Process Mining Techniques. *Knowledge and Information Systems*, 21(3):267–297, 2009.

[132] Ana Karla A. de Medeiros, Anton J.M.M. Weijters, and Wil M. P. van der Aalst. Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.

[133] Jan Mendling, Hajo A. Reijers, and Wil M. P. van der Aalst. Seven Process Modeling Guidelines (7PMG). *Information and Software Technology*, 52(2):127–136, 2010.

[134] Michael Merz, Daniel Moldt, K. Müller, and Winfried Lamersdorf. Workflow Modelling and Execution with Coloured Petri Nets in COSM. In *Workshop on Applications of Petri Nets to Protocols, Proceedings 16th International Conference on Application and Theory of Petri Nets*, pages 1–12, 1995.

[135] Thomas P. Minka. Expectation Propagation for Approximate Bayesian Inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

[136] Michael K. Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. PhD thesis, University of California, Los Angeles, 1981.

[137] Michael K. Molloy. Performance Analysis Using Stochastic Petri Nets. *Computers, IEEE Transactions on*, 100(9):913–917, 1982.

[138] Gordon E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 8(8):114–117, April 1965.

[139] Hamid R. Motahari-Nezhad, Regis Saint-Paul, Fabio Casati, and Boualem Benatallah. Event Correlation for Process Discovery from Web Service Interaction Logs. *The VLDB Journal–The International Journal on Very Large Data Bases*, 20(3):417–444, 2011.

[140] Tadao Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[141] Kevin P. Murphy. The Bayes Net Toolbox for Matlab. In *Interface'01*, volume 33 of *Computing Science and Statistics*, pages 1024–1034, 2001.

[142] Kevin P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, 2002.

[143] Kreshnik Musaraj, Tetsuya Yoshida, Florian Daniel, Mohand-Said Hacid, Fabio Casati, and Boualem Benatallah. Message Correlation and Web Service Protocol Mining from Inaccurate Logs. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 259–266. IEEE, 2010.

[144] Michael zur Mühlen and Jan Recker. How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In *CAiSE '08*, volume 5074 of *Lecture Notes in Computer Science*, pages 465–479, Berlin, Heidelberg, 2008. Springer-Verlag.

[145] Michael zur Mühlen and Robert Shapiro. Business Process Analytics. In Jan vom Brocke and Michael Rosemann, editors, *Handbook on Business Process Management*, volume 2, pages 137–157. Springer, 2010.

[146] Richard Müller and Andreas Rogge-Solti. BPMN for Healthcare Processes. In Daniel Eichhorn, Agnes Koschmider, and Huayu Zhang, editors, *Proceedings of the 3rd Central-European Workshop on Services and their Composition, ZEUS 2011, Karlsruhe, Germany, February 21–22, 2011*, volume 705 of *CEUR Workshop Proceedings*, pages 65–72. CEUR-WS.org, 2011.

[147] Joyce Nakatumba and Wil M. P. van der Aalst. Analyzing Resource Behavior Using Process Mining. In *Business Process Management Workshops*, pages 69–80. Springer, 2010.

[148] Stephen Natkin. *Reseaux de Petri Stochastiques*. PhD thesis, Conservatoire National des Arts et Métiers (CNAM), Paris, France, 1980.

[149] Radford M. Neal. Slice sampling. *The Annals of Statistics*, 31(3):705–741, 2003.

[150] Object Management Group (OMG). Business Process Model and Notation (BPMN) 2.0 Specification, January 2011.

[151] Object Management Group (OMG). Unified Modeling Language, Infrastructure, V2.4.1, nov 2011.

[152] Fariborz Y. Partovi and Jonathan S. Burton. Timing of monitoring and control of CPM projects. *Engineering Management, IEEE Transactions on*, 40(1):68–75, 1993.

[153] Judea Pearl. Fusion, Propagation, and Structuring in Belief Networks. *Artificial intelligence*, 29(3):241–288, 1986.

[154] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[155] Judea Pearl. *Causality*. Cambridge University Press, 2000.

[156] Carlos Pedrinaci, John Domingue, and Ana Karla Alves de Medeiros. A Core Ontology for Business Process Analysis. In *The Semantic Web: Research and Applications*, pages 49–64. Springer, 2008.

[157] Ricardo Pérez-Castillo, Barbara Weber, Ignacio G.-R. de Guzmán, Mario Piattini, and Jakob Pinggera. Assessing Event Correlation in Non-Process-Aware Information Systems. *Software & Systems Modeling*, pages 1–23, 2012.

[158] Ricardo Pérez-Castillo, Barbara Weber, Jakob Pinggera, Stefan Zugal, Ignacio G.-R. de Guzmán, and Mario Piattini. Generating Event Logs from Non-Process-Aware Systems Enabling Business Process Mining. *Enterprise Information Systems*, 5(3):301–335, 2011.

[159] Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Technische Hochschule Darmstadt, 1962.

[160] Avi Pfeffer. Functional Specification of Probabilistic Process Models. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, volume 20, page 663. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[161] Antonio Puliafito, Miklós Telek, and Kishor S. Trivedi. The Evolution of Stochastic Petri Nets. In *Proceedings of World Congress on Systems Simulation, Singapore*, pages 3–15, September 1977.

[162] Chander Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, Massachusetts Institute of Technology, 1973.

[163] Tzvi Raz and Erdal Erel. Optimal timing of Project Control Points. *European Journal of Operational Research*, 127(2000):252–261, 2000.

[164] James Reason. *Human Error*. Cambridge University Press, 1990.

[165] Manfred Reichert and Peter Dadam. ADEPT flex–Supporting Synamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[166] Hajo A. Reijers. *Design and Control of Workflow Processes : Business Process Management for the Service Industry*. PhD thesis, Technische Universiteit Eindhoven, 2002.

[167] Hajo A Reijers and S. Liman Mansar. Best Practices in Business Process Redesign: An Overview and Qualitative Evaluation of Successful Redesign Heuristics. *Omega*, 33(4):283–306, 2005.

[168] Wolfgang Reisig. *Petrinetze—Modellierungstechnik, Analysemethoden, Fallstudien*. Vieweg+Teubner Verlag, 2010.

[169] Barry Render, Ralph M. Stair Jr., and Michael E. Hanna. *Quantitative Analysis for Management*. Pearson Education Inc., 11th edition, 2012.

[170] Bernhard Rochell and Norbert Roeder. Drg—das neue krankenhausvergütungssystem für deutschland. *Der Urologe, Ausgabe A*, 42(4):471–484, 2003.

[171] Andreas Rogge-Solti, Wil M. P. van der Aalst, and Mathias Weske. Discovering Stochastic Petri Nets with Arbitrary Delay Distributions From Event Logs. In *Business Process Management Workshops*. Springer, 2014 (to appear).

[172] Andreas Rogge-Solti, Nico Herzberg, and Luise Pufahl. Selecting Event Monitoring Points for Optimal Prediction Quality. In Stefanie Rinderle-Ma and Mathias Weske, editors, *EMISA*, volume 206 of *GI-Edition - Lecture Notes in Informatics (LNI)*, pages 39–52, 2012.

[173] Andreas Rogge-Solti, Ronny S. Mans, Wil M. P. van der Aalst, and Mathias Weske. Improving Documentation by Repairing Event Logs. In Janis Grabis, Marite Kirikova, Jelena Zdravkovic, and Janis Stirna, editors, *The Practice of Enterprise Modeling*, volume 165 of *Lecture Notes in Business Information Processing*, pages 129–144. Springer Berlin Heidelberg, 2013.

[174] Andreas Rogge-Solti, Ronny S. Mans, Wil M. P. van der Aalst, and Mathias Weske. Repairing Event Logs Using Stochastic Process Models. Technical Report 78, Hasso Plattner Institute at the University of Potsdam, Germany, 2013.

[175] Andreas Rogge-Solti, Ronny S. Mans, Wil M. P. van der Aalst, and Mathias Weske. Repairing Event Logs Using Timed Process Models. In Yan T. Demey and Hervé Panetto, editors, *OTM 2013 Workshops*, pages 705–708, Heidelberg, 2013. Springer.

[176] Andreas Rogge-Solti and Mathias Weske. Enabling Probabilistic Process Monitoring in Non-automated Environments. In *Enterprise, Business-Process and Information Systems Modeling*, volume 113 of *Lecture Notes in*

*Business Information Processing*, pages 226–240. Springer Berlin Heidelberg, 2012.

[177] 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 *Lecture Notes in Computer Science*, pages 389–403. Springer, Heidelberg, 2013.

[178] Jerome Rolia and Vidar Vetland. Parameter Estimation for Performance Models of Distributed Application Systems. In *Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research*, CASCON '95, pages 54–64. IBM Press, 1995.

[179] Marcello La Rosa and Pnina Soffer, editors. *Business Process Management Workshops - BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012.*, volume 132 of *Lecture Notes in Business Information Processing*. Springer, 2013.

[180] Arturo Rosenblueth and Norbert Wiener. The Role of Models in Science. *Philosophy of Science*, 12(4):316–321, 1945.

[181] Anne Rozinat and Wil M. P. van der Aalst. Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In *Business Process Management Workshops*, pages 163–176. Springer, 2006.

[182] Anne Rozinat and Wil M. P. van der Aalst. Conformance Checking of Processes based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.

[183] Anne Rozinat, Ronny S. Mans, Minseok Song, and Wil M. P. van der Aalst. Discovering simulation models. *Information Systems*, 34(3):305–327, May 2009.

[184] Anne Rozinat, Moe Thandar Wynn, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Colin J. Fidge. Workflow Simulation for Operational Decision Support. *Data & Knowledge Engineering*, 68(9):834–850, 2009.

[185] Szabolcs Rozsnyai, Aleksander Slominski, and Geetika T.Ch Lakshmanan. Automated Correlation Discovery for Semi-Structured Business Processes. In *IEEE 27th International Conference on Data Engineering Workshops (ICDEW) 2011*, pages 261–266. IEEE, 2011.

[186] Donald B. Rubin. Inference and Missing Data. *Biometrika*, 63(3):581–592, 1976.

[187] Donald B. Rubin. *Multiple Imputation for Nonresponse in Surveys*. Wiley, 1987.

[188] Khodakaram Salimifard and Mike Wright. Petri Net-Based Modelling of Workflow Systems: An Overview. *European Journal of Operational Research*, 134(3):664–676, 2001.

[189] Joseph L. Schafer and John W. Graham. Missing Data: Our View of the State of the Art. *Psychological methods*, 7(2):147–177, 2002.

[190] Michael Schiffers. Behandlung eines Synchronisationproblems mit gefärbten Petri Netzen. Diplomarbeit, Universität Bonn, 1977.

[191] Helen Schonenberg, Natalia Sidorova, Wil M. P. van der Aalst, and Kees van Hee. History-Dependent Stochastic Petri Nets. In Amir Pnueli, Irina Virbitskaite, and Andrei Voronkov, editors, *Perspectives of Systems Informatics*, volume 5947 of *Lecture Notes in Computer Science*, pages 366–379. Springer Berlin Heidelberg, 2010.

[192] Bruce R. Silver. *BPMN Method and Style: With BPMN Implementer's Guide*. Cody-Cassidy Press Aptos, 2011.

[193] Sweta Sneha and Upkar Varshney. Enabling Ubiquitous Patient Monitoring: Model, Decision Protocols, Opportunities and Challenges. *Decision Support Systems*, 46(3):606–619, 2009.

[194] Jan A. Snyman. *Practical Mathematical Optimization – An IIntroduction to Basic Optimization Theory and ClasClass and New Gradient-Based Algorithms*, volume 97 of *Applied Optimization*. Springer, 2005.

[195] Moira Stewart, Judith B. Brown, W. Wayne Weston, Ian R. McWhinney, Carol L. McWilliam, and Thomas R. Freeman. *Patient-Centered Medicine: Transforming the Clinical Method*. Radcliffe Medical Press, 2nd edition, 2003.

[196] David P. Strum, Jerrold H. May, and Luis G. Vargas. Modeling the uncertainty of surgical procedure times: comparison of log-normal and normal models. *Anesthesiology*, 92(4):1160–7, April 2000.

[197] Fred J. W. Symons. *Modeling and Analysis of Communication Protocols Using Numerical Petri Nets*. PhD thesis, University of Essex, Great Britain, 1978.

[198] Manoj Thomas, Richard Redmond, Victoria Yoon, and Rahul Singh. A semantic approach to monitor business process. *Communications of the ACM*, 48(12):55–59, 2005.

[199] Kishor S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, volume 2002. John Wiley & Sons, 2 edition, 2002.

[200] Alexey Tsymbal. The Problem of Concept Drift: Definitions and Related Work. Technical report, Computer Science Department, Trinity College Dublin, 2004.

[201] H. M. W. (Eric) Verbeek. BPI Challenge 2012: The Transition System Case. In Marcello Rosa and Pnina Soffer, editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 225–226. Springer Berlin Heidelberg, 2013.

[202] H. M. W. (Eric) Verbeek, Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. XES, xESame, and ProM 6. In Pnina Soffer and Erik Proper, editors, *Information Systems Evolution – CAiSE Forum 2010, Hammamet, Tunisia, June 7-9, 2010, Selected Extended Papers*, pages 60–75. Springer Berlin Heidelberg, 2011.

[203] William Wu-Shyong Wei. *Time Series Analysis*. Addison-Wesley Redwood City, California, 1994.

[204] Burton A. Weisbrod. The Health Care Quadrilemma: An Essay on Technological Change, Insurance, Quality of Care, and Cost Containment. *Journal of Economic Literature*, 29(2):523–552, 1991.

[205] Michael P. Wellman. Fundamental Concepts of Qualitative Probabilistic Networks. *Artificial Intelligence*, 44(3):257–303, 1990.

[206] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, second edition, 2012.

[207] Workflow Management Coalition (WFMC). Terminology and Glossary. Technical Report WFMC-TC-1011, Workflow Management Coalition, Brussels, 1996.

[208] Gerhard Widmer and Miroslav Kubat. Learning in the Presence of Concept Drift and Hidden Contexts. *Machine learning*, 23(1):69–101, 1996.

[209] Graham N. Wilkinson. Estimation of Missing Values for the Analysis of Incomplete Data. *Biometrics*, 14(2):257–286, 1958.

[210] Glynn Winskel. Petri Nets, Algebras, Morphisms, and Compositionality. *Information and Computation*, 72(3):197–238, 1987.

[211] Andreas Wombacher and Maria-Eugenia Iacob. Estimating the Processing Time of Process Instances in Semi-structured Processes–A Case Study. In *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, pages 368–375. IEEE, 2012.

[212] Andreas Wombacher and Maria-Eugenia lacob. Start Time and Duration Distribution Estimation in Semi-Structured Processes. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1403–1409, New York, NY, USA, 2013. ACM.

[213] Ying Yang, Xindong Wu, and Xingquan Zhu. Mining in Anticipation for Concept Change: Proactive-Reactive Prediction in Data Streams. *Data mining and knowledge discovery*, 13(3):261–289, 2006.

[214] Liangzhao Zeng, Christoph Lingenfelder, Hui Lei, and Henry Chang. Event-Driven Quality of Service Prediction. In *Service-Oriented Computing–ICSOC 2008*, pages 147–161. Springer, 2008.

[215] Chengxiang Zhai and John Lafferty. A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '01, pages 334–342. ACM, 2001.

[216] Lizheng Zhang, Weijen Chen, Yuhen Hu, and Charlie C. Chen. Statistical Static Timing Analysis with Conditional Linear MAX/MIN Approximation and Extended Canonical Timing Model. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(6):1183–1191, 2006.

[217] Huiyuan Zheng, Jian Yang, Weiliang Zhao, and Athman Bouguettaya. QoS Analysis for Web Service Compositions Based on Probabilistic QoS. In *Service-Oriented Computing 2011*, pages 47–61. Springer, 2011.

[218] Armin Zimmermann. Modeling and Evaluation of Stochastic Petri Nets with TimeNET 4.1. In *Performance Evaluation Methodologies and Tools (VALUETOOLS), 2012 6th International Conference on*, pages 54–63. IEEE, 2012.

[219] Armin Zimmermann, Jörn Freiheit, and Günter Hommel. Discrete Time Stochastic Petri Nets for the Modeling and Evaluation of Real-Time Systems. In *Parallel and Distributed Processing Symposium., Proceedings 15th International*, pages 1069–1074, 2001.

[220] Indrė Žliobaitė. Learning Under Concept Drift: An Overview. Technical report, Faculty of Mathematics and Informatics, Vilnius University, Lithuania, 2009.