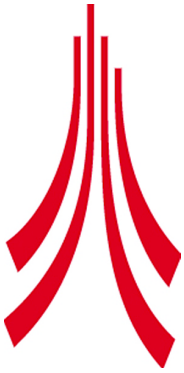
Descriptive Business Process Models at Run-time



David Redlich, B.Sc., M.Sc.(Hons)

School of Computing and Communications

Lancaster University

This dissertation is submitted for the degree of

Doctor of Philosophy

March 2018

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 100,000 words including footnotes and appendices but excluding the bibliography.

David Redlich, B.Sc., M.Sc.(Hons)
March 2018

Acknowledgements

First and foremost, I would like to thank my academic and industrial supervisors, Gordon Blair, Awais Rashid, and Wasif Gilani, for their guidance and feedback as well as their continuous motivation and support throughout the course of my PhD studies. I would have never seen the end of it without your help.

I would also like to extend my gratitude towards SAP in general and my managers in particular: Phil Taylor and Wasif Gilani (yes, you are mentioned twice), who have always supported me on my way towards thesis submission and provided helpful guidance about how to survive in the industrial as well as the academic world.

My colleagues, Thomas Molka, Marc Drobek, Mykola Galushka, Ulrich Winkler, and Mathias Fritzsche, deserve also a very big thanks. They made work a fun and enjoyable place to be, but more importantly were always there to bounce ideas back and forth. This helped my research progress enormously.

Last but not least, I would like to thank and acknowledge the undergraduate students who contributed some of the implementation based on my ideas and also tremendously helped to showcase the work with their unflagging effort to prepare demonstrations and develop fancy visualisations: Stephanie Platz, Thomas Hornoff, Tobias Proske, Robert Fritzsche, Max Bothe, Stefan Bunk, Johannes Jasper, Balazs Pinter, and Julien Bergner.

Abstract

Today's competitive markets require organisations to react proactively to changes in their environment if financial and legal consequences are to be avoided. Since business processes are elementary parts of modern organisations they are also required to efficiently adapt to these changes in quick and flexible ways. This requirement demands a more dynamic handling of business processes, i.e. treating business processes as run-time artefacts rather than design-time artefacts. One general approach to address this problem is provided by the community of *models@run.time*, which promotes methodologies concerned with self-adaptive systems where models reflect the system's current state at any point in time and allow immediate reasoning and adaptation mechanisms. However, in contrast to common self-adaptive systems the domain of business processes features two additional challenges: (i) a bigger than usual abstraction gap between the business process models and the actual run-time information of the enterprise system and (ii) the possibility of run-time deviations from the planned models. Developing an understanding of such processes is a crucial necessity in order to optimise business processes and dynamically adapt to changing demands.

This thesis explores the potential of adopting and enhancing principles and mechanisms from the models@run.time domain to the business process domain for the purpose of run-time reasoning, i.e. investigating the potential role of *Descriptive Business Process Models at Run-time* (DBPMRTs) in the business process management domain. The DBPMRT is a model *describing* the enterprise system at run-time and thus enabling higher-level reasoning on the as-is state. Along with the specification of the DBPMRT, algorithms and an overall framework are proposed to establish and maintain a causal link from the enterprise system to the DBPMRT at run-time. Furthermore, it is shown that proactive higher-level reasoning on a DBPMRT in the form of performance prediction allows for more accurate results. By taking these steps the thesis addresses general challenges of business process management, e.g. dealing with frequently changing processes and shortening the business process life cycle. At the same time this thesis contributes to research in models@run.time by providing a complex real-world use case as well as a reference approach for dealing with volatile models@run.time of a higher abstraction level.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Problem Domain

The success of modern organisations depends on the efficiency and performance of their employed *Business Processes* (BPs). These processes dictate the flow of work for high-level business functions to achieve business goals. Prominent examples of business processes are *Order-to-Cash*, *Accounts-Receivable*, or *Procure-to-Pay*. They represent fundamental parts of many organisations and are considered to be *"...the most valuable corporate asset"* [270]. Business processes can reside on different levels of an organisation, e.g. operational vs. strategic [122], and can have different degrees of automation, e.g. fully automated vs. manual execution.

In order to deal with increasing complexity and to efficiently manage business processes, IT-based solutions have been harnessed. This development led to the rise of *Business Process Management* (BPM), as an IT-related discipline. In fact, BPM is an interdisciplinary subject of "theory in practice" adopting a variety of paradigms and methodologies from computer science, management theory, philosophy, mathematics, and linguistic, just to name a few [95, 122, 194]. Business processes and their management are supported by (Enterprise) *Business Process Management Systems* (BPMS's). They are implementations of so called "application and integration middleware", e.g. *SAP Netweaver BPM* [287], *IBM WebSphere* [102], *Intalio BPMS Designer* [103].

In BPM, a business process is represented by a *Business Process Model* specifying different aspects of the process like activities, sequence, performance, resources, and roles. In industry, many standards for business process models exist, e.g. *Business Process Model and Notation* (BPMN) [168], *Business Process Execution Language* (BPEL) [161], or *Business Process Query Language* (BPQL) [164]. Because of the inter-disciplinary nature of BPM, each of these model standards are also subject to influence from non-IT disciplines and thus were designed for one or more different purposes, e.g. comprehension, communication, execution, or diagnosis. With regard to their time of validity in relation to the enterprise system those standards can be categorised into two different types (see Figure 1.1): *a-priori* and *a-posteriori* business process models.

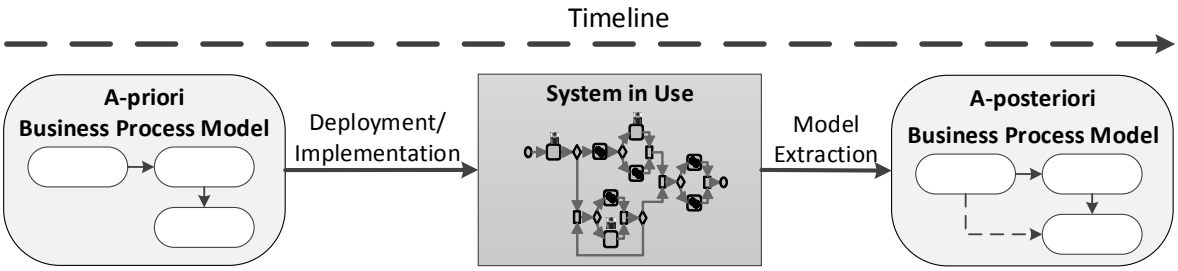


Fig. 1.1 The Two Common Types of Business Process Models

A-priori business process models are design-time models, i.e. business processes are designed or documented *before* execution to specify or pre-analyse the processes in an organisation. This is either done informally via a document listing and describing the steps and their execution sequence or they are modelled via design-time languages. The most prominent business process model languages were developed to build human-readable design-time models with the focus on aspects like interoperability, or being a basis for reliable communication between different stakeholders [47].

In contrast, a-posteriori business process models are extracted *after* execution for diagnosis purposes to reflect the *real* behaviour of the observed system. This kind of retrospective analysis of business processes based on execution logs is called *Process Mining* (PM) [231]. PM becomes necessary if no design-time model is available or the actual business process execution deviates from the designed model. Such a deviation is common for all non-fully automated business processes and can be caused by, for instance, an under-specification of the designed model, a reaction to exceptional circumstances, or an evolution of the process during run-time [251].

1.2 Problem Statement

Today's businesses need to survive in a highly competitive environment which comes with strong requirements with regards to fast adaptation and optimisation. Because late, inadequate, or incorrect actions can lead to Service Level Agreement (SLA) violations, causing financial and legal consequences, the ability to react proactively to changes is a crucial feature for successful organisations. Since business processes are driving today's modern organisations they are ultimately required to adapt and evolve continuously in order to meet demands and constraints inflicted by internal or external sources [203], e.g. changing market contexts, customer requirements, or business imperatives. One example of such a dynamic organisation is a hospital, in which the treatment processes have to be flexible and quickly modifiable, based on the qualifications of the current staff and patients' demands, e.g. emergencies, epidemics. Another example is a retail company which is exposed to a lot of market pressure and has to quickly adapt to changing customer demands and shortages of resources and/or goods.

In order to optimise business processes and adapt to changing demands, develop-

ing an understanding of the deployed processes is the first immediate and crucial necessity. This information enables a business analyst to make insightful decisions and answer questions related to *compliance*, *performance*, or *sustainability*, e.g.: How are my business functions executed in reality? How do my business processes generally perform? How will this performance change in the future? What are current and expected bottlenecks of the deployed business processes?

However, discovering and understanding the current condition of deployed processes is a difficult task: *Firstly*, business processes are often either not documented or poorly documented. *Secondly*, modern organisations and their deployed processes are of an increasingly dynamic nature. As a result, even if documented, the actual execution of business processes usually deviates from the modelled design-time business processes due to changes occurring during run-time. A-priori business process models are, therefore, not accurate enough to be used as information sources for decision making. Process mining algorithms, on the other hand, extract a-posteriori models from historic execution logs and are, therefore, too static to respond to the dynamic nature of flexible business processes, i.e. these models can already be outdated at the time of extraction.

In conclusion, both types of business process models, a-priori and a-posteriori models, exhibit a chronological distance to the running system. Furthermore, due to process evolution and exceptional circumstances during run-time a factual difference between a design-time model and the business process in the running system is often observable. Both the identified chronological distances and factual differences imply a lack of causality between business process models and the system executing the business processes. This makes reasoning on this information difficult and susceptible to errors.

1.3 Overall Goal and Objectives

The overall goal of this thesis is to mitigate the lack of causality between system and business process model in real-time environments in order to promote a more dynamic handling of business processes and enable higher-level reasoning on run-time data.

A general approach to achieve this goal is provided by the *models@run.time* (MRT) community, which is concerned with model-driven mechanisms and principles to purposefully (self-)reflect on an associated system during run-time [16]. This thesis explores the adoption of principles and mechanisms from the MRT domain to the business process domain for the purpose of run-time reasoning, thus, investigating the potential role of *Descriptive Business Process Models at Run-time* (DBPMRT). As opposed to a-priori or a-posteriori models, DBPMRT is a model of a business process *describing* the enterprise system *while* in use and therefore represents a new type of model in the business process management domain. To achieve this goal, the following objectives have to be approached and are the subject of this thesis (conceptually depicted in Figure 1.2):

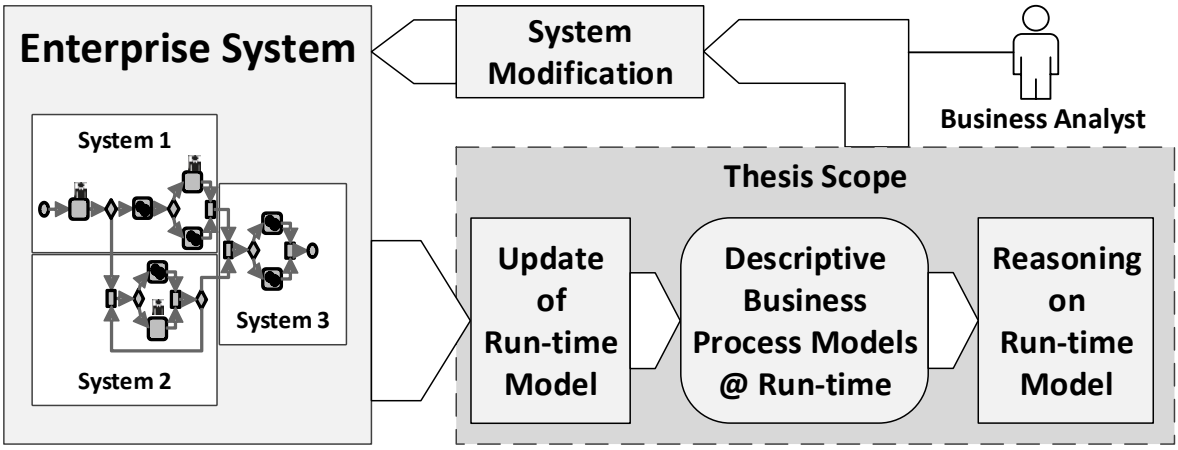


Fig. 1.2 Scope and Objectives of Thesis

Specification of Descriptive Business Process Model at Run-time (DBPMRT):
DBPMRT has to deal with additional concerns as opposed to common business process models, such as capturing status information of the enterprise system's processes. One of the main objectives is to identify these run-time characteristics and elaborate a DBPMRT specification that allows for expressing these run-time characteristics.

Establishing the Link from Business Process Management System to DBPMRT:
Enterprise systems like BPMSs produce execution events indicating a change of the system's internal state. Since they represent fine-granule state transitions, they are on a different abstraction level than business processes. One objective is to infer from this low level event data higher level BP information in a real-time environment, i.e. every change in the system should be *causally* and *timely* reflected in the associated run-time representation (i.e. DPBPMRT). Due to run-time constraints and the abstract nature of business process models this objective is particularly challenging as the connection from system to DPBPMRT is difficult to maintain.

Reasoning on DBPMRT:
The potential benefit of reasoning on a DBPMRT is that the system's complete state information can be utilised to carry out high-level analyses for decision support, e.g. performance prediction, detection of bottlenecks or SLA violations. One of the objectives of this thesis is to provide decision support functionality based directly on the current descriptive state of the business process and thus show that reasoning mechanisms built on the introduced concept of DBPMRT have the potential to yield more accurate results.

1.4 Research Questions

These objectives raise a number of research questions that are addressed by this thesis:

1. *How can the target system (BPMS) be effectively represented at run-time?*
2. *To what extent can a causal link from BPMS to BP model be established given that event data and BP model conform to different levels of abstraction?*
3. *To what degree can a causal link from BPMS to BP model be maintained in a dynamic run-time environment?*
4. *Can the quality of run-time reasoning be improved by utilising DBPMRTs?*

In a broader sense this thesis also provides a partial answer, i.e. in the form of a use case, to the question:

5. *How and to what extent can a target system emitting low-level events be causally and timely reflected by a run-time model of a higher abstraction level?*

1.5 Research Strategy

In order to achieve the main goal and objectives the following research strategy has been adopted (see Figure 1.3):

1. Requirement Collection: The research carried out in the context of this thesis is driven by the requirements of real-life use cases provided by SAP, a large business software vendor, and TIMBUS, an EU funded research project concerned with the preservation of business processes [84]. These use cases drove the need for DBPMRT and continued to shape the agenda for each of the involved objectives throughout the research process, e.g. run-time constraints of algorithms, what information and what types of reasoning are relevant for business analysts in order to proactively initiate business process modifications and/or preservation.
2. State of the Art Review: In this phase, a state of the art and industry survey in the domains of business process management and run-time models as well as related disciplines has been carried out.
3. Determining the potential role of DBPMRT: By evaluating the state of the art and reflecting on the nature of descriptive business process run-time models, the objectives and the respective individual challenges that need addressing have been determined.
4. Designing a framework for DBPMRT: In this phase, a framework and the involved run-time models for DBPMRT were designed responding to the topic's objectives and complying to the constraints and requirements imposed by the industrial use cases.

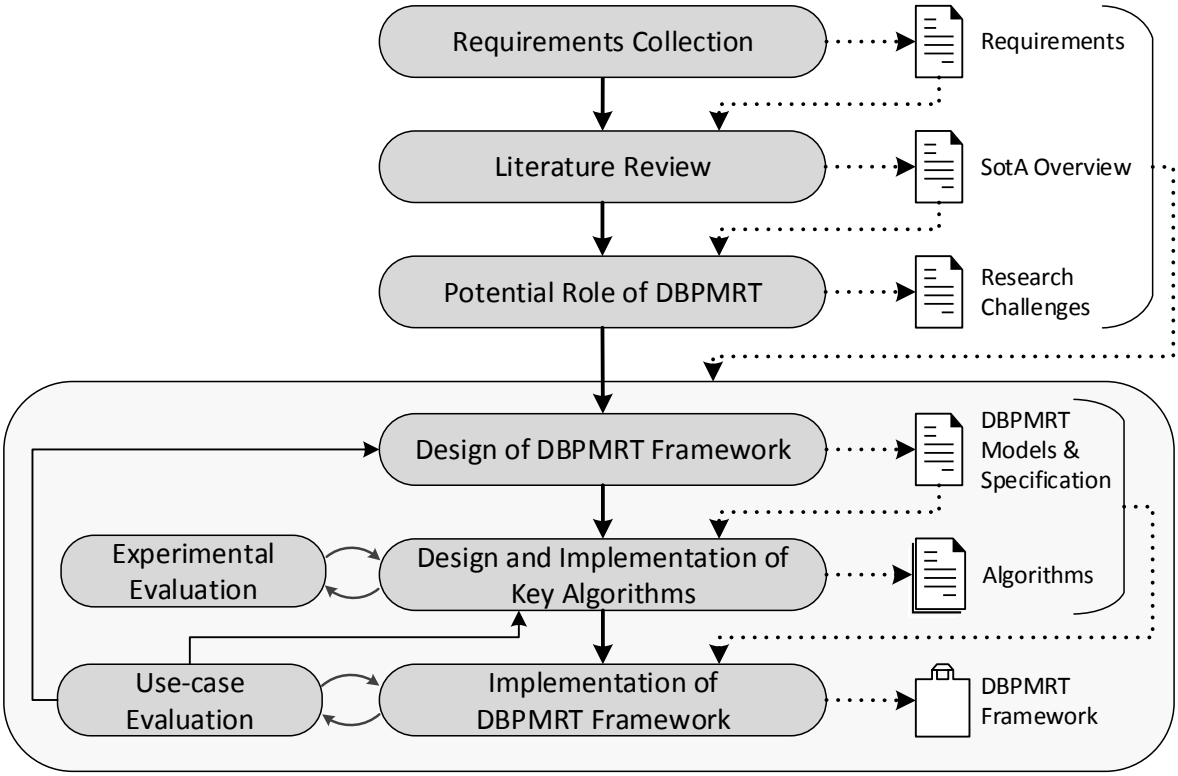


Fig. 1.3 Schematic View of Research Strategy

5. Addressing of the individual challenges: Key algorithms have been implemented in order to address individual challenges of each of the different objectives. The quality of the key algorithms have been quantitatively evaluated against other relevant state-of-the-art algorithms in experiments based on artificial and real-life data.

6. Prototypical implementation of the DBPMRT framework: In this final phase, a proof of concept for DBPMRT framework has been carried out. This was achieved by implementing a DBPMRT solution compliant with the framework specifications which was then utilised and evaluated in the context of industrial use cases provided by SAP and the TIMBUS project.

1.6 Research Philosophy & Research Method

Depending on the problem to be investigated and previous advances on that subject the appropriate scientific approach needs to be chosen. Computer Science is influenced by many other subjects, e.g. Logic, Mathematics, Physics, Chemistry, Biology, and Psychology, and thus employs a large variety of different scientific approaches [54, 59]. Two fundamentally different paradigms exist: *interpretivism* and *positivism* [42]: (1) Research conducted via interpretivism employs *qualitative* evaluations that are based on subjective data and follow a rather argumentative methodology; (2) Research conducted in the positivism fashion is based on *quantitative* evaluations, i.e. it involves collecting objective (and mostly numerical) data (measurements) and evaluating it statistically.

The two fundamental paradigms also apply to research in the specific field of Software Engineering. For instance, when performance is of relevance for the research topic the more traditional paradigm of positivism is the obvious choice [1]. In contrast, when such an empirical measure is not clearly evident - as is often the case in Software Engineering - the research follows the interpretive paradigm. In the latter case the nature of scientific questions asked is more important than the empirical method [87, 177], i.e. research must ask questions that can be falsified by observations [108]. The implication is that the scientific quality of research in Software Engineering improves if it *"...is successfully developed into practical applications (i.e. as a methodology, technique or tool)."* [8] For this purpose *Case Studies* are often employed - they are suitable to describe, understand, and explain the research subject as well as validating the results [8, 59].

Concretely, this thesis is concerned with Software Engineering *for BPM*, i.e. the resulting artefacts are *Information Systems*. According to Hevner et al. in [97], the traditional research paradigm debates (positivism vs. interpretivism) stem from natural sciences and are based on the assumption that *"... somewhere some truth exists and somehow that truth can be extracted, explicated, and codified"*. They argue that most of the research in Information Science can be characterised by two other paradigms: behavioural science and design science [97]: (1) *"the behavioural-science paradigm seeks to develop and verify theories that describe or predict human or organizational behavior"*, while (2) the *"design-science paradigm seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts"*. The design science paradigm is closely related with the process of engineering and the sciences of the artificial [216], i.e. it does not seek to find the one truth but rather what is effective. Hence, research in design science should be aligned with real world production experience, i.e. case studies [97, 143].

The research conducted as part of this thesis is concerned with the analysis of business processes, i.e. of organisations and the underlying Information Systems. However, while the goal is to discover and predict human and organisational behaviour, the contributions of this thesis are artefacts for the *general purpose* of analysing the behaviour but not for the analysis of a *particular* behavioural reality. *It therefore follows the design science paradigm.* In order to answer the research questions in accordance with the design science paradigm the effectiveness of the artefacts is *evaluated through case studies: (1) Multiple different event logs (real world and artificial) to evaluate the quality of the algorithm artefacts and (2) A case study to evaluate correctness of the model specifications as well as the overall advanced reasoning framework, i.e. the DBPMRT framework.* When regarded in the context of the traditional research paradigms of positivism and interpretivism, *this work follows a hybrid approach: It employs the positivist paradigm when a quantitative analysis is performable, i.e. when algorithm artefacts are concerned and when established quality metrics exist, but will also employ interpretivism methodologies when only a argumentative reasoning is possible*, e.g. when behavioural models are extracted for which no quantitative measure exist.

1.7 Main Contributions

The foremost and most important contribution of this thesis is the adoption of principles and theories of the models@run.time community to the abstraction level of business processes. By taking this step the thesis addresses general challenges of business process management [122, 123, 214], i.e. dealing with frequently changing processes and shortening the business process life cycle, and contributes to research in models@run.time by providing a valid use case as well as further requirements for models@run.time of a higher abstraction level. In particular, this thesis yields the following contributions (including relevant paper publications as lead author):

1. Identification of run-time characteristics of BPs and composition of a DBPMRT reference specification:
The first contribution of this thesis is the identification of run-time characteristics of BPs and the design of a set of DBPMRT meta-models capturing a descriptive reflection of the BP system. The DBPMRTs comprise information of important perspectives of a business process, i.e. control-flow, resources, performance, instance states, and are able to represent different levels of change, i.e. a step of the process was executed vs. the entire process has changed. Part of the findings (with regards to need for representing the different levels of change) have been published in [181]. Additionally, in order to deal with the challenge of differing abstraction levels (low level event data vs. high level BP model) an approach for distinguishing between two different types of run-time models has been proposed in [184]: *computer-oriented* footprint models vs. *human-oriented* BP models.

2. Key algorithms to establish a causal link from BPMS to BP model at run-time:

- A novel algorithm to establish a causal link between event data and a descriptive BP control-flow model: The *Constructs Competition Miner* (CCM) published in [185] follows a top-down approach to discover block-structured process models from event logs possibly containing exceptional and contradictory behaviour.
- Modification of the statically operating CCM algorithm in order to work in a real-time setting to detect changes in the BP control-flow while processing a stream of BP execution events: The *Dynamic Constructs Competition Miner* (DCCM) is published in [186]. A second evaluation of the DCCM on the eHealth use case "DrugFusion" (provided by the TIMBUS project) is published in [182].
- A dynamic concept for the event-based update of higher abstraction levels of the model (e.g. change in performance, control-flow, etc.) including concepts like "time-dependent ageing" and "discrete ageing". The different ageing approaches have been published in the context of the control-flow perspective of a BP in [187].

- A smart and time-efficient algorithm to capture the fine-grained instance state of an enterprise system. As of the time of writing this algorithm has not yet been individually published.

3. DBPMRT Framework:
The third contribution is the design and proof of concept implementation for a generic framework for Descriptive Business Process Models at Run-time, enabling (1) automated monitoring of an enterprise system by capturing the state of the system in a descriptive run-time model and (2) real-time reasoning based on the descriptive run-time model. The concepts and component specifications of the framework have been published in [184].

4. Real-time Reasoning Approach based on DBPMRT and Simulation:
This thesis' contribution includes a generic conceptual approach for predictive reasoning on DBPMRTs. Since a DBPMRT comprises current and historical data about all important perspectives and all levels of change it can be utilised by a BP simulation to predict future behaviour. Consequently, already existing reasoning algorithms, e.g. performance prediction, detection of future bottlenecks, what-if analysis, can be (re-)used to enable *proactive* decision support on future states of the system. Parts of the concept have been published in [183] and [188]. In the context of this thesis the predictive reasoning concept is introduced and evaluated for the use case of performance prediction.

Additionally, the author has contributed in varying degrees to publications which are related to the main contributions but not entirely covered within the scope of this thesis. This includes publications in the fields of (design-time) BP simulations [57, 58, 178, 189, 286], BP discovery based on genetic algorithms [147, 148], BP conformance [146], digital preservation [84], risk management [83], and mechanisms for MRT in self-adaptive software [12]. To further promote the topic of DBPM@RT, many of the thesis' concepts have been applied and evaluated in the context of the multiple industrial business use case scenarios made available within the European research project TIMBUS as well as in internal SAP projects. During this process a number of demonstration tools have been developed and knowledge transfer projects with SAP's productisation departments have been initiated. A complete collection of all publications the author produced or contributed to during the course of the PhD studies is presented in Annex A.

1.8 Thesis Roadmap

Figure 1.4 gives an overview and a proposed reading flow of the chapters. The thesis consists of three general parts: In *Part I* introduction, background, and current state of the art for the related research disciplines is presented. *Part II* individually addresses the objectives of this thesis as well as describes and evaluates the main contributions. In *Part III* these contributions are qualitatively assessed and future work is presented. The remainder of the thesis is structured as follows:

Chapter 2: State-of-the-Art Business Processes and Model-driven Engineering, comprises a discussion of state of the industry and state of the art for relevant disciplines in the business process and model-driven engineering domain. This includes fundamental information about business processes and business process management as well as models and model-driven engineering. Furthermore, the current state-of-the-art in the topics of process discovery, business process analysis, business process flexibility, and models at run-time are discussed and reviewed.

Chapter 3: Gap Analysis of Descriptive Models at Run-time in Business Process Management, discusses the consolidation of the previously introduced research disciplines and the resulting challenges. In this context gaps in the current state-of-the-art work is identified for the three individual challenges of this thesis.

Chapter 4: Descriptive Business Process Models at Run-time, addresses the first objective of this thesis by specifying a reference language for DBPMRTs. The specification is qualitatively evaluated against pre-specified requirements inferred from the challenges.

Chapter 5: Establishment of Casual Connection, describes how the objective of monitoring a system under study whilst maintaining an accurate run-time model is addressed, i.e. how a causal link from system to DBPMRT can be established and maintained in a dynamic run-time environment. It furthermore presents the algorithms and components that are involved in the proposed concept. The concept and algorithms are quantitatively and qualitatively evaluated.

Chapter 6: Reasoning on DBPMRTs, presents a concept to harness the run-time information in a DBPMRT for high-level reasoning, i.e. in particular, short-term performance prediction via simulation. Furthermore, a case scenario is introduced which is the basis of the evaluation of the reasoning concept along with the overall DBPMRT concept in the context of short-term performance prediction.

Chapter 7: Conclusion and Future Research, concludes the thesis by revisiting and answering the research questions as well as highlighting the main contributions of this thesis. Additionally, an agenda for future research is proposed.

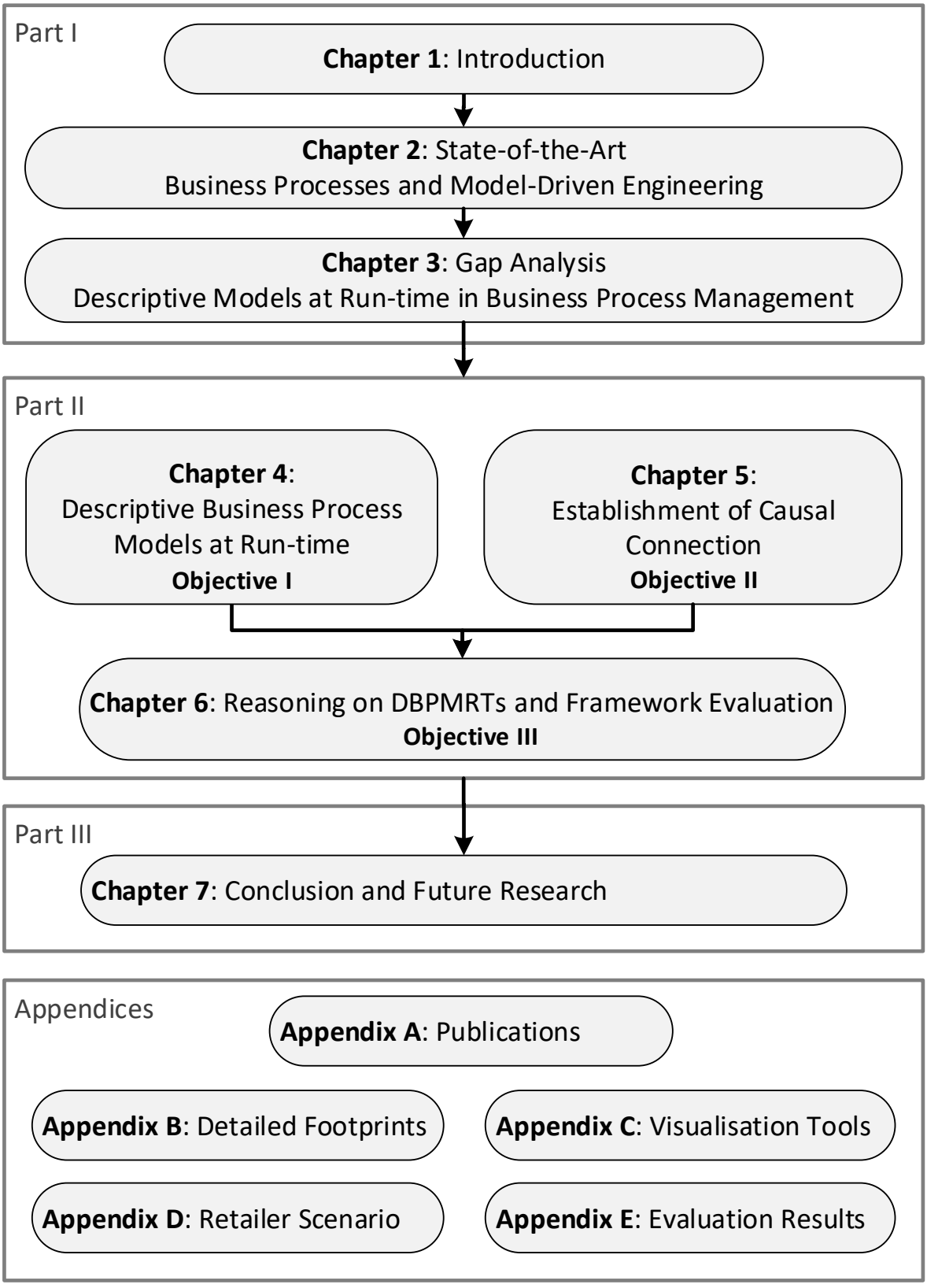


Fig. 1.4 Structure and Chapters of the Thesis

Chapter 2

State of the Art: Business Processes and Model-Driven Engineering

In this chapter state of the art in research and industry for different topics of the business process domain and Model-Driven Engineering (MDE) domain are discussed. In the beginning a basic introduction to business processes is provided (Section 2.1), which is followed by a discussion about the *management*, *lifecycle*, *standards*, and *analysis* of business processes (Section 2.2). This is followed by Section 2.3 in which recent work in the field of process flexibility, i.e. changes applied during enactment of a process, is presented. Furthermore, two relevant topics with regards to descriptive and predictive analysis of business processes are discussed, namely *Process Discovery* (Section 2.4) and *Performance Decision Support for Business Processes* (Section 2.5). Then relevant fundamentals as well as state of the art work is discussed for the MDE domain in general and Models at Run-time (MRT) domain in particular. First in Section 2.6, an introduction to the goals, principles, and definitions of MDE is provided. This is followed by a state of the art analysis of current advances in the domain of models at run-time in Section 2.7, notably, with regards to objectives, architecture, model composition, techniques, and other characteristics of MRT in current literature. A particular focus throughout this analysis will be on aspects of descriptive models at run-time. Figure 2.1 shows the overview and the individual dependencies between the described sections.

2.1 Business Processes

Processes accompany every human venture, from simply booking a holiday to manufacturing a car. In a similar way a business organisation is driven by its so-called *business processes*: Its elementary tasks have to be carried out in certain ways in order to achieve business goals and meet predefined objectives. These tasks, their order of execution, and the resources to perform them are usually defined in one or more business processes. In [122] business processes are defined as *"...a series or network of value-added activities, performed by their relevant roles or collaborators, to purposefully achieve the common*

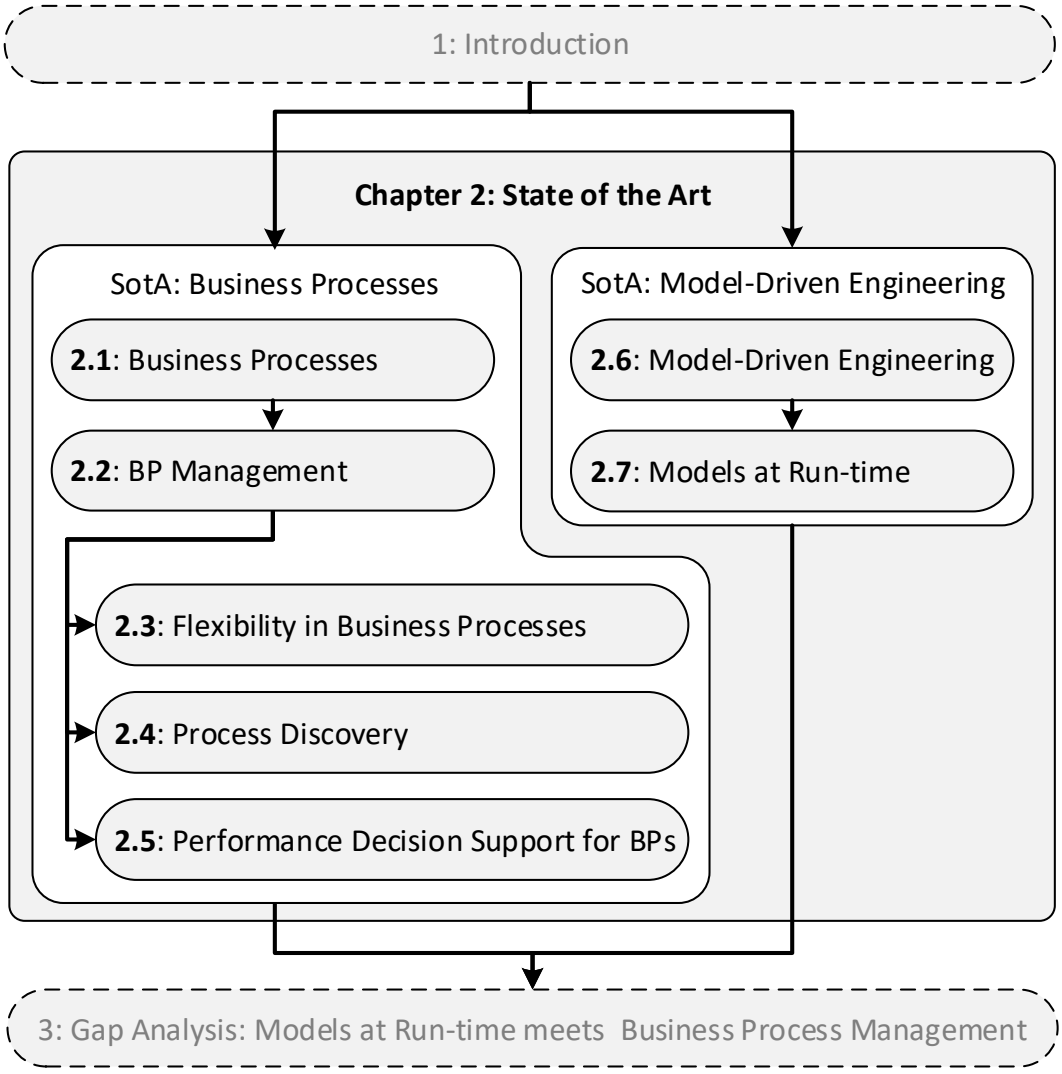


Fig. 2.1 Outline of Chapter 2

business goal." Figure 2.2 depicts an example business process: Online-Order-Processing. It is specified by a set of essential elements, further described in the following:

Start and End Event are required parts of every business process, respectively indicating its entry (start event) and exit points (end event), i.e. they represent instantiation and termination of a process.

Activities are process steps which abstract the execution of actual work. Three different types of activities exist:

1. *Automated Activities* are service-like work tasks that can be immediately performed when requested, i.e. the allocated resource may execute multiple automated activities concurrently at the same time without a noticeable delay. For instance, "Credit Check" in the example process is such a service-like activity and thus can be immediately and automatically executed by a "Credit Server" which can process multiple service requests at the same time and without delay.

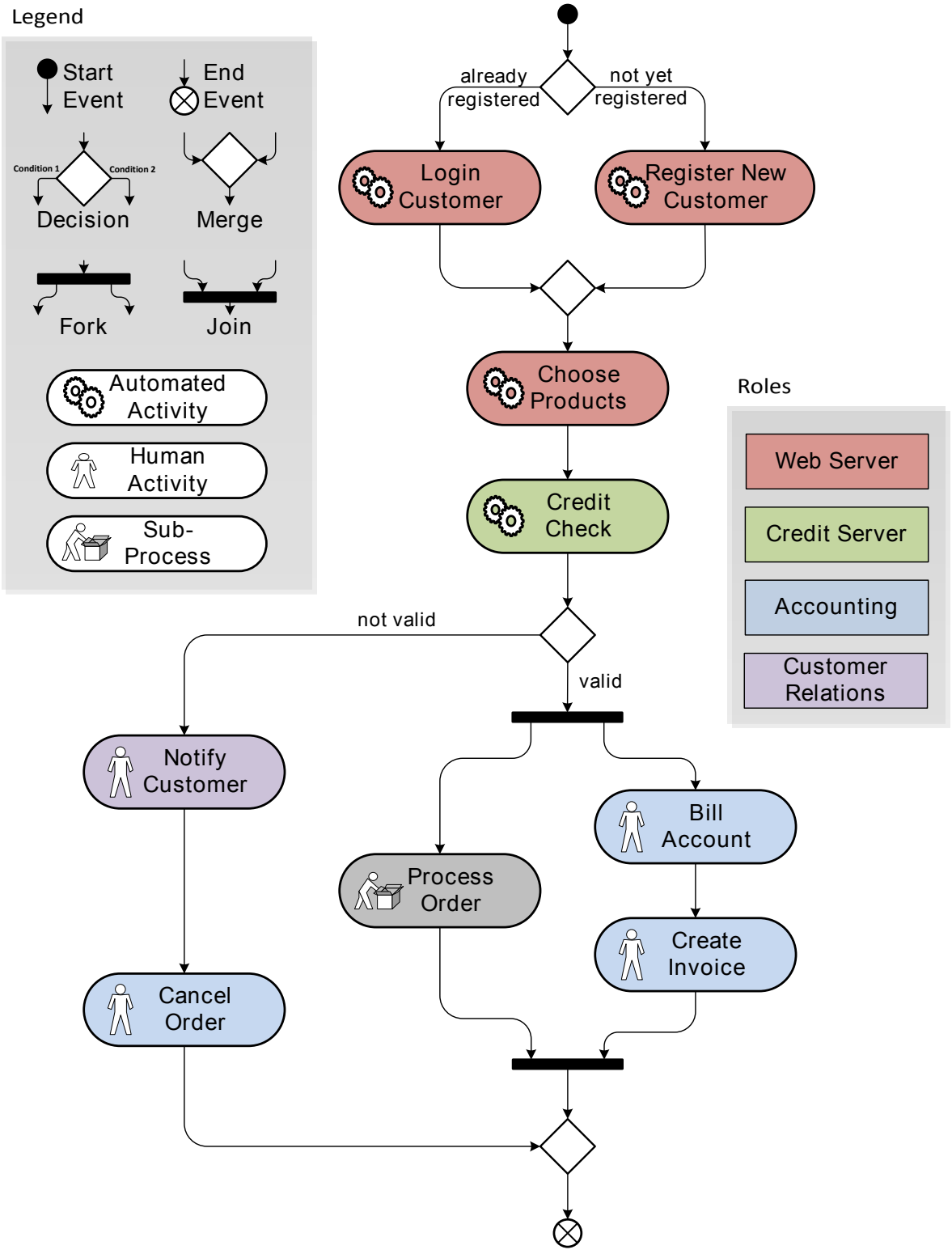


Fig. 2.2 Online-Order-Processing Business Process

2. *Human Activities*, in contrast, do require a resource to actively perform the work task. These "active" resources can only execute one activity at the time and are associated with one or more *Roles* which specify their areas of responsibility. An activity has to be executed by a resource that is qualified to perform the associated task, i.e. only resources of a specific role are possible. If currently no resource is available the task cannot be executed and the process execution cannot continue until the required resource becomes available. "Jane Doe", for instance, is a resource that has the roles "Accounting" and "Human Resources": If he is currently not occupied she can perform all tasks that require these roles, e.g. "Bill Account" and "Create Invoice" but cannot perform "Notify Customer".

3. *Sub-Process* is a non-atomic activity[1] that abstracts a processes of a lower hierarchical level, e.g. the "Process Order" step from the Online-Order-Processing example represents such a sub-process and has its own specification. The usage of sub-processes makes it possible to abstract and modularise business processes.

Fork and Join elements are used to direct the workflow of the process. A fork represents a splitting of the current process execution into two or more parallel paths, i.e. each of the paths can be performed in parallel without dictating the order of execution. Its counterpart, the Join represents a synchronisation point for the process, i.e. the execution only proceeds once all of the incoming parallel paths are completed. This means for the example that the process can only continue if activities "Create Invoice" and "Process Order" were completed.

Decision and Merge elements are used to direct conditional flows in a process. A Decision is semantically equal to an exclusive choice between two or more target paths, a certain path will be followed if a corresponding condition is true. For instance, the decision in the example process after "Credit Check" represents a choice between either continuing or cancelling the order depending on whether or not the credit check was successful, i.e. valid or not valid. Its counterpart, a Merge, is the join element which is enabled once one of the exclusive source paths is completed. The merge element does not synchronise. Note, that also loop behaviour can be constructed with the help of these two elements.

This list only contains common elements that are shared by all relevant business process specifications, i.e. a process can also consists of elements which are not listed above. However, these elements are usually language-specific and/or of a different abstraction level.

In the business process domain a *process type* is a particular type of process with a defined business goal, e.g. Online-Order-Processing has the goal of high customer satisfaction which translates to reducing the end-to-end processing time. A process type is

[1] automated and human activities are atomic activities since they can be not split up further

represented by a particular *process schema* which is captured in a *business process model* specifying the behavioural information of a business process like activities, ordering, resources [168]. Figure 2.2 shows a particular process schema of Online-Order-Processing. A process type may be represented by more than one process schema expressing different versions or evolution steps of this type. Furthermore, a *process instance* is a single execution/occurrence of the business process, i.e. a particular sequence of executed activities in order to process a *work item*.

Furthermore, it is common to distinguish between different perspectives of a business process. For the scope of this thesis the following three perspectives are of importance:

- *Control-flow Perspective*: describes the execution order of process activities with the help of control elements [233], e.g. start events, forks, and decisions (see Figure 2.2). Since it captures the behavioural information of a process it is sometimes also referred to as *behavioural perspective*, e.g. [190].

- *Resource Perspective*: describes the classification of resources (e.g. people, systems) into roles [233] and which of these entities are in charge of executing the activities of the process. In Figure 2.2, only the roles (on the right) and their respective associations to the process activities are shown (via different colours). Since this perspective captures the organisational structure of the process it is also referred to as *organisational perspective*, e.g. [190, 233].

- *Performance Perspective*: describes the performance of a business process, which includes for instance information about the execution time of an activity, or how often the process has been initiated, or with which probability certain paths are chosen. This perspective is of a retrospective nature and not modelled at design-time but rather extracted after or while execution. Since the majority of performance information is related to timing and frequencies, it is sometimes also referred to as *time perspective*, e.g. [233].

Additional perspectives exist, e.g. (1) *Functional Perspective*: defines what a process has to do, i.e. the goals of the process [190], and (2) *Data Perspective* (or Case Perspective [233]) which is concerned with specific information that is associated with a single instantiation of the process, e.g. items ordered, money to be transferred. However, those perspectives are highly dependent on use case and implementation of the process and are therefore not easily generalised. This thesis focuses on an autonomous and general solution that is independent from external input and as such not use case specific, i.e. it focuses on the perspectives that can be generalised: Control-flow, Resource, and Performance.

2.2 Business Process Management

The increasing complexity and importance of business processes initiated the development of Business Process Management (BPM) as an IT-related research area. BPM is a cross-discipline subject of "theory in practice" adopting a number of different concepts and methodologies from various domains such as computer science, management theory, philosophy, linguistics, and mathematics [122]. Perhaps because of its cross-disciplinary nature, even after three decades, there are many duplicate and contradictory publications trying to clarify definition and scope of basic BPM terminology [122], e.g. business process vs. workflow, BPM vs. Workflow Management (WfM) vs. Business Process Reengineering (BPR).

Business Process Management (BPM) is, however, considered to be the next step after the workflow wave of the nineties [246]. Originating from the domain of workflow management, many definitions of BPM are based on workflow terminology. A *Workflow Management System* (WfMS) is defined as: *"A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications."* [125]. In a similar fashion BPM is defined as: *"Supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents, and other sources of information."* [246]. Software systems that support the execution and general management of operational business processes are referred to as Business Process Management Systems or Business Process Management Suites (BPMS's) [123]. Popular examples of BPMS are *SAP Netweaver BPM* [287], *IBM WebSphere* [102], *Intalio BPMS Designer* [103].

Part of the complete BPM definition is the BPM life-cycle. It originates from the standard development life cycle and consists of 4 stages (see Figure 2.3) [246]:

1. **Process Design** - In this stage, business processes are modelled for the BPMS.
2. **System Configuration** - This stage configures the BPMS and the underlying system infrastructure (e.g., synchronisation of roles).
3. **Process Enactment** - The modelled business processes are deployed and executed in a BPMS.
4. **Diagnosis** - With analysis and monitoring tools, the BPM analyst can identify bottlenecks and improve the business processes.

The viewpoint of van der Aalst et al. is that WfM covers only process design, system configuration, and process enactment, but BPM also includes the diagnosis phase to complete the BPM lifecycle [246]. This viewpoint makes WfM a logical subset of BPM. According to [122] *"...many BPMS are still very much workflow management systems (WfMS) and have not yet matured in the support of the BPM diagnosis."* However, recently the

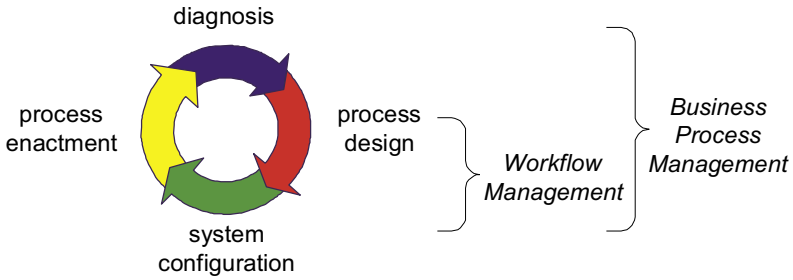


Fig. 2.3 BPM life-cycle: Difference between Workflow Management and Business Process Management [246]

diagnosis phase started to gain more attention which is reflected in the high number of publications in the fields of *Business Process Analysis* (BPA) (see Section 2.2.3) in general and the sub-topics of *Process Discovery* (see Section 2.4) and *Process Intelligence* (PI) (see Section 2.2.3 and Section 2.5) in particular. Other definitions of the BPM lifecycle exist but basically centre around the same or similar four lifecycle steps, e.g. Configure, Execute, Analyse, and Decide in [70].

A more industrial viewpoint on BPM and WfM is provided by Gartner [98]: *"Business process management (BPM) is a process-oriented management discipline. It is not a technology. Workflow is a flow management technology found in business process management suites (BPMS's) and other product categories."* Here BPM is a management discipline which is supported by WfM as a technology.

To put BPM terminology into one coherent picture, understanding the nested relationship of BPM theory (e.g., Pi Calculus [145] and Petri Nets [172]), BPM standards (e.g., Business Process Model and Notation (BPMN) [168] or Business Process Execution Language (BPEL) [161]), and BPM systems (e.g. SAP Netweaver BPM [287] or Intalio BPMS Designer [103]) is important: BPM standards are based on established BPM theory and eventually adopted into software, i.e. BPMSs [123]. This nested relationship is depicted in Figure 2.4.

2.2.1 Business Process Languages and Modelling Standards

Today's large-scale business processes, which potentially span across multiple organisations, are usually collaboratively executed on a set of different BPMS instances, each complying to its own standards. As of 2007, there were more than 10 formal groups creating BPM standards, which produced at least 7 specifications for business process modelling [291]. Since then, industry has adopted many of these standards and specifications and new ones have been developed. However, the diversity of standards and the fact that especially the industrial standards are mostly only informally specified makes it difficult to generalise towards one business process representation. In order to get an overview about the state of the art for business process modelling standards it makes sense to cat-

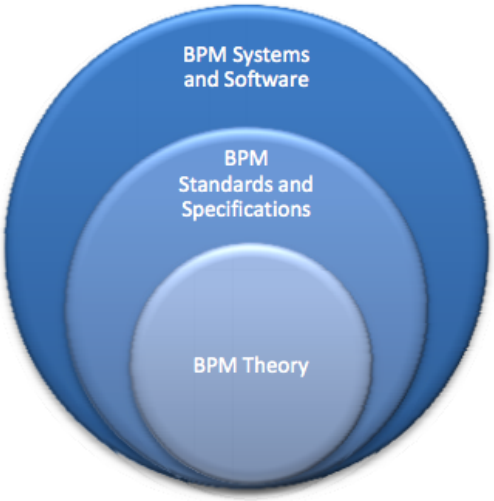


Fig. 2.4 Nested Relationship between BPM Theory, Standards, and Systems [123]

egorise them into groups with similar functions and characteristics [123]. Many of the standards address at least one of the phases of the BPM life cycle. For this reason Ko et al. suggest a separation of features found in existing standards into four different types of standards [123]:

1. **Graphical Standards** allow users to express information flow, decision points, and roles for business processes in a diagrammatic way. Standards of this type correspond to the design phase of the BPM life cycle and are usually the easiest to understand, i.e. human-readable. Prominent examples of graphical standards are Business Process Model and Notation (BPMN) [168], Event-driven Process Chains (EPC) [207], and activity diagrams of Unified Modelling Language (UML) [165].

2. **Execution Standards** are code-like and enable business processes to be deployed in a BPMS. Standards of this type correspond to the enactment phase of the BPM lifecycle. The most prominent example is Business Process Execution Language (BPEL) (sometimes also called Web Service Business Execution Language (WS-BPEL)) [161].

3. **Interchange Standards** are used to translate graphical standards to execution standards and exchange business process models between BPMS's [144]. One of the reasons these standards became necessary was the fragmented BPM landscape. Two prominent examples of interchange standards exist: Business Process Definition Metamodel (BPDM) [167] and XML Process Definition Language (XPDL) [289].

4. **Diagnosis Standards** provide monitoring capabilities. These standards are to support audit trails, real-time business process information, trend analysis, bottleneck identification, etc. Examples are initiatives of Object Management Group: Business Process Runtime Interface (BPRI) [166] and Business Process Query Language (BPQL) [164]. Though, both of the projects have not produced a standard (as of yet).

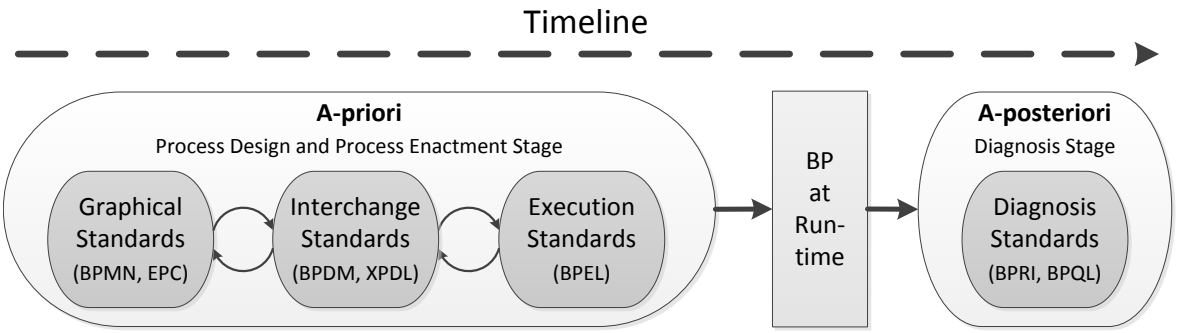


Fig. 2.5 Common Kinds of Business Process Model Standards

Most of the existing standards dealing with modelling languages can be assigned to one of these types. Of course, this a simplified view and standards exist which can be mapped to more than just one type, e.g. Yet Another Workflow Language (YAWL) [245] can be regarded as graphical and execution standard, or BPEL which can have a graphical representation, too.

The previous classification can be further generalised: Considering only the relation of each of the standards to the system with regards to their time of validity, practically only two different types remain (see Figure 2.5):

- **A-priori model** - Business process models at design-time: In this case business processes are documented *before* execution. This documentation can range from formal, i.e. using a language defined by one of the BP Modelling Standards (e.g. BPEL), to very informal, i.e. defining different design-aspects of the process in a concept-like manner (e.g. word document). Basically, every prominent language that addresses the enactment phase or one of the preceding is considered a-priori model, e.g. BPMN or BPEL, and focuses on design-time aspects like communication and interoperability [47].

- **A-posteriori model** - In practice business process models are often required to be extracted *after* execution to reflect the real behaviour of a process as part of the diagnosis phase of the BPM life cycle. This extraction is necessary if no design-time model exists or a run-time deviation from the designed model has occurred, e.g. caused by an under-specification, a reaction to exceptional circumstances, or an evolution of the process during run-time [251]. This static a-posteriori analysis of business processes based on process execution logs is called Process Mining [231] and is discussed in more detail in Section 2.2.3. Since no standards for the diagnosis phase exist as of yet, a-posteriori models usually do not conform to BP-domain specific languages but to formal languages of BPM theory, e.g. Petri Nets [172].

2.2.2 Process Execution Event Logs

A computer-aided execution of a business processes via BPMS's produces *process execution event logs* (or short *event logs*) containing transactional details for each event occurring during the execution. These events are usually of a simple nature and often only comprise raw and direct information describing the state transition, but not the state of the whole system [233]. However, since the execution can span over multiple BPMS's each individual event log can look very different. Differences can occur on many dimensions, e.g. event granularity (activity level events vs. instance level events), perspective support (whether resource, instance or other information is provided or not). Due to the differences in the various log formats, efforts have been made to provide an extensible and general log format, e.g. *Mining XML* (MXML) [256], its successor *eXtensible Event Stream* (XES) [90, 258], or *Business Process Analytics Format* (BPAF) [288, 293]. In the case of an execution of collaborative BPs spanning multiple different BPMS, events have to be captured, filtered, and merged to one seamless log conforming to such a generic log format before further analysis (discussed in Section 2.2.3) is carried out.

Events captured in a process execution log which conforms to a general log format like XES or BPAF have the following characteristics [90, 293] (Table 2.1 contains an excerpt of an execution event log resulting from Process in Figure 2.2):

Table 2.1 Excerpt of Execution Event Log resulting from Process in Figure 2.2

Event ID	Timestamp	Instance ID	Name	Resource	Lifecycle
296213	Thu Dec 16 11:47:27 GMT 1999	9520	Bill Account	Eli Findmer	complete
296214	Thu Dec 16 11:47:27 GMT 1999	9538	Bill Account	Eli Findmer	assign
296215	Thu Dec 16 11:47:27 GMT 1999	9520	Create Invoice	-	schedule
296216	Thu Dec 16 11:48:20 GMT 1999	9486	Create Invoice	Mia Larson	complete
296217	Thu Dec 16 11:48:20 GMT 1999	9472	Create Invoice	Mia Larson	assign
296218	Thu Dec 16 11:48:20 GMT 1999	9486	End Online Order	-	trace_end
296221	Thu Dec 16 11:48:41 GMT 1999	9468	Choose Products	Webserver_Instance	complete
296222	Thu Dec 16 11:48:41 GMT 1999	9468	Credit Check	-	schedule
296223	Thu Dec 16 11:48:50 GMT 1999	9468	Credit Check	CreditServer_Instance	complete
296224	Thu Dec 16 11:48:50 GMT 1999	9468	Sub Process Order	-	schedule
296227	Thu Dec 16 11:48:50 GMT 1999	9468	Bill Account	-	schedule
296233	Thu Dec 16 11:49:02 GMT 1999	9473	Sub Process Order	-	complete
296236	Thu Dec 16 11:49:32 GMT 1999	9540	Sub Process Order	-	complete
296237	Thu Dec 16 11:50:11 GMT 1999	9519	Bill Account	Viktor Charmical	complete
296238	Thu Dec 16 11:50:11 GMT 1999	9579	Create Invoice	Viktor Charmical	assign
296239	Thu Dec 16 11:50:11 GMT 1999	9519	Create Invoice	-	schedule
296251	Thu Dec 16 11:52:57 GMT 1999	9487	Create Invoice	Chuck Tchahovsky	assign
296259	Thu Dec 16 11:53:22 GMT 1999	9548	Login in known customer	Webserver_Instance	complete
296260	Thu Dec 16 11:53:22 GMT 1999	9548	Choose Products	-	schedule
296264	Thu Dec 16 11:54:38 GMT 1999	9546	Registering new customer	Webserver_Instance	complete
296265	Thu Dec 16 11:54:38 GMT 1999	9546	Choose Products	-	schedule
296266	Thu Dec 16 11:56:43 GMT 1999	9570	Start Online Order	-	triggered
296267	Thu Dec 16 11:56:43 GMT 1999	9570	Login in known customer	-	schedule
296270	Thu Dec 16 11:57:33 GMT 1999	9466	Start Online Order	-	triggered
296271	Thu Dec 16 11:57:33 GMT 1999	9466	Registering new customer	-	schedule
296274	Thu Dec 16 11:59:24 GMT 1999	9569	Choose Products	Webserver_Instance	complete
296275	Thu Dec 16 11:59:24 GMT 1999	9569	Credit Check	-	schedule
296276	Thu Dec 16 11:59:35 GMT 1999	9569	Credit Check	CreditServer_Instance	complete
296277	Thu Dec 16 11:59:35 GMT 1999	9569	Sub Process Order	-	schedule
296280	Thu Dec 16 11:59:35 GMT 1999	9569	Bill Account	-	schedule
296281	Thu Dec 16 11:59:36 GMT 1999	9557	Bill Account	Julia Walker	complete
296282	Thu Dec 16 11:59:36 GMT 1999	9474	Bill Account	Julia Walker	assign
296283	Thu Dec 16 11:59:36 GMT 1999	9557	Create Invoice	-	schedule

- *Event ID*: The event's unique identifier.
- *Timestamp*: The date and time, at which the event has occurred.
- *Process Definition ID*: If events of different processes are captured in a log, each event requires a unique identifier to the corresponding process definition the event belongs to. In Table 2.1 no Process Definition ID is displayed since the example log only contains events from one process definition.
- *Instance ID*: A Log is a record of the execution details of the instances of one or more business processes. Each event in the log represents a state transition that occurred during the execution of a process instance, i.e. all events in a log corresponding to the same instance make up a *trace*, which is also sometimes called *case* in literature (e.g. in [233]). The *Instance ID* (or *Trace ID*, or *Case ID*) is the unique identifier that specifies to which process instance this event belongs to. Note, that an Instance ID only has to be unique in the universe of one process definition, i.e. traces of different business processes may have the same Instance ID. In fact, it can occur that traces span multiple business processes for which a shared identifier is required to analyse inter-process dependencies.
- *Name*: Represents the generally understood name of the event, e.g. the name of the executed activity represented by the event [90]. This event name is assumed to be a unique identifier of the BP element in the process definition this event is associated with and should not allow for duplicates. Events indicating the start and end of a process may be part of the log but may have a customised "name" entry that is not a direct reference to a BP element in the process definition (see Table 2.1 Event IDs 296218, 296266, and 296270).
- *Resource*: The name of the resource that triggered the event [90]. Similar to the name attribute it is assumed to be a unique identifier to the resource associated with the event, e.g. a unique employee id as opposed to the employee's name. If no resource is associated with this event, this value is empty.
- *Lifecycle*: Events in most logs capture state transition on the activity level, i.e. events represent activity lifecycle transitions. In these cases the processing of an activity is recorded by more than one event and the corresponding transitions are captured by a lifecycle attribute. This is illustrated in Table 2.1 in which Event IDs 296222 and 296223 represent two different activity lifecycles of the same activity and instance. MXML and XES propose a general and unified activity lifecycle based on state transitions as displayed in Figure 2.6 [90, 256]. In contrast, the BPAF activity lifecycle is based on states and not on state transitions but proposes a mapping to the MXML/XES state transition system [288, 293]. However, it consists of more states (13 vs. 7 states) and thus allows for more behaviour. Note, that the event formats of many BPMS may record behaviour less complex than enforced

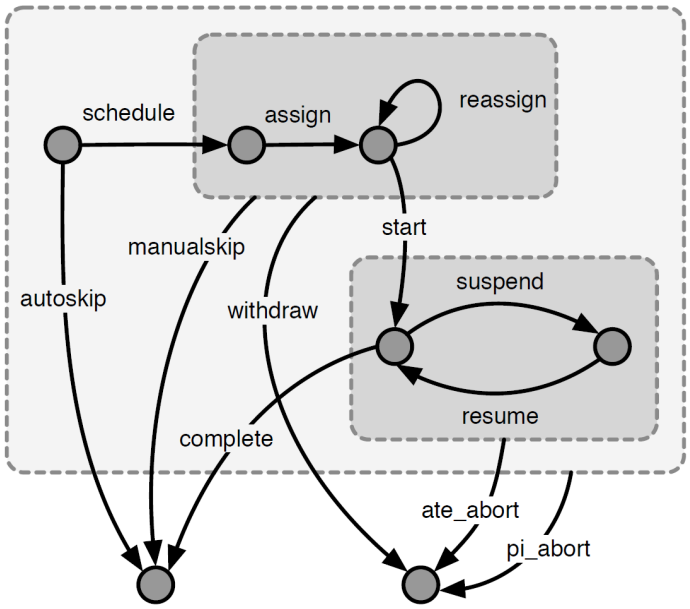


Fig. 2.6 Lifecycle transitions in MXML and XES [90, 256]

by the activity lifecycle of MXML/XES. For instance, a successful completion of an activity execution in the MXML/XES format requires a minimum of four events: $schedule \rightarrow assign \rightarrow start \rightarrow complete$ whereas the successful completion of the automated activity "Credit Check" is expressed by only two events in Table 2.1 (Event IDs 296222 and 296223): $schedule \rightarrow complete$. Some logs may even only contain one single event per processed activity, e.g. when the task has been completed.

Since general standards like XES and BPAF are extensible, more information can be contained in the events. For instance, additional organisational data (e.g. the role the resource is associated with), additional cost data (e.g. cost and currency of the activity execution), or additional transactional data (e.g. items of an order) [90].

2.2.3 Overview of Business Process Analyses

Business Process Analysis/Analytics *"... provides process participants, decision makers, and related stakeholders with insight about the efficiency and effectiveness of organizational processes"* [292]. As such it is an important part of Business Process Management since it enables a continuous improvement of planned or employed business processes. Many techniques and research topics addressing the analysis of running or modelled BPs exist and are sometimes difficult to differentiate, e.g. Process Mining vs. Business Activity Monitoring vs. Process Intelligence. Through the help of an extended BPM Lifecycle adopted from [251] those types of BPAs can be put into context: Figure 2.7 shows the extended BPM Lifecycle and relevant types of business process analyses in relation to it. One of the differences in comparison to the traditional BPM lifecyle in Figure 2.3 is that instead of one process lifecycle it consists of three lifecycles. The main lifecycle is still

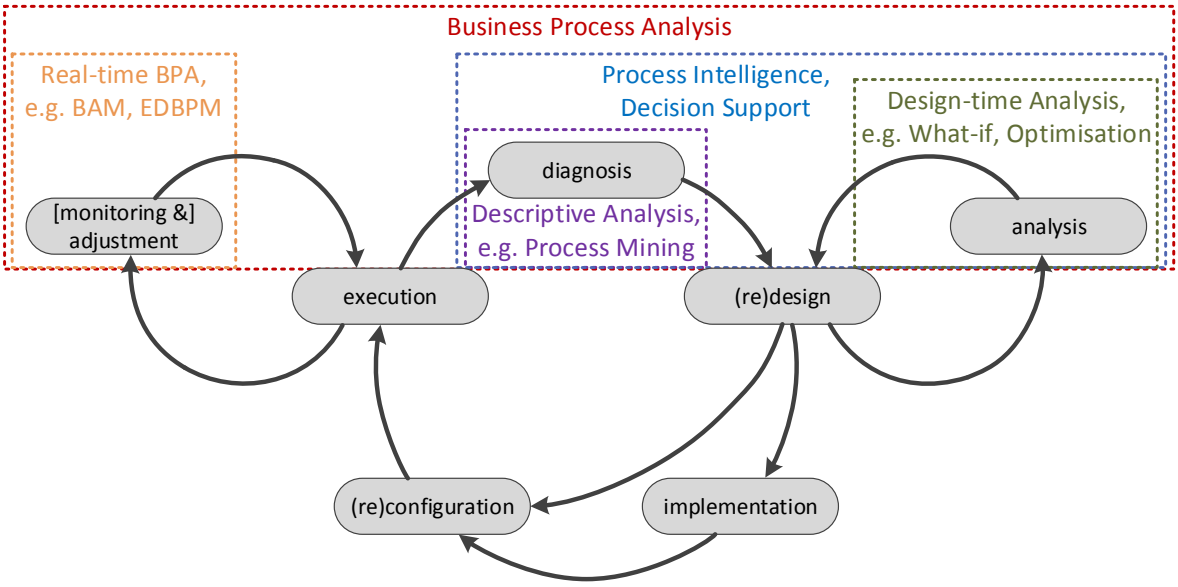


Fig. 2.7 Extended BPM Lifecycle [251] and different types of BPA in relation

dominant and only slightly changed[2] (Figure 2.7 - centre cycle). Additionally, two more cycles have been introduced: One which allows for monitoring[3] and adjustments of the process during run-time (left cycle) and one that describes the iterative procedure employing predictive methods while designing the process (right cycle). The following two types of Business Process Analyses can be distinguished:

Process Intelligence / Decision Support

Decision Support and Process Intelligence are two terms that basically describe the same type of business process analyses: These are analyses supporting stakeholders and business analyst in making decisions about the business process through providing them with additional computed information of diverse nature, e.g. performance of the business process or the predicted behaviour of a planned control-flow. This information enables the stakeholders and analysts to obtain more insight into the behaviour of the actual or future process execution and allows for insightful decision making when designing a new iteration of the process. Note, that a human entity is explicitly involved in the diagnosis and analysis phases of the lifecycle, i.e. the configuration, execution, and evaluation of the process intelligence analyses, as well as the eventual decision making are all human-guided. Whereas the definition of Process Intelligence by Muehlen et al. mostly focusses on forecasting future behaviour [292], the definition by Aalst et al. includes descriptive techniques and even automated run-time monitoring[4] [251]. Here, the term Process Intelligence is defined as an umbrella term for all business process related high-

[2] now an optional implementation step is included acknowledging differentiation between design/graphical and execution standards for BPs (see Section 2.2.1)

[3] the term "monitoring" was added to the "adjustment" step of the original extended lifecycle from [251]

[4] This is mostly due to the fact that Aalst et al. do not distinguish between an ex-post analysis of an event log and immediate event processing - for them both constitute an analysis at run-time [230]

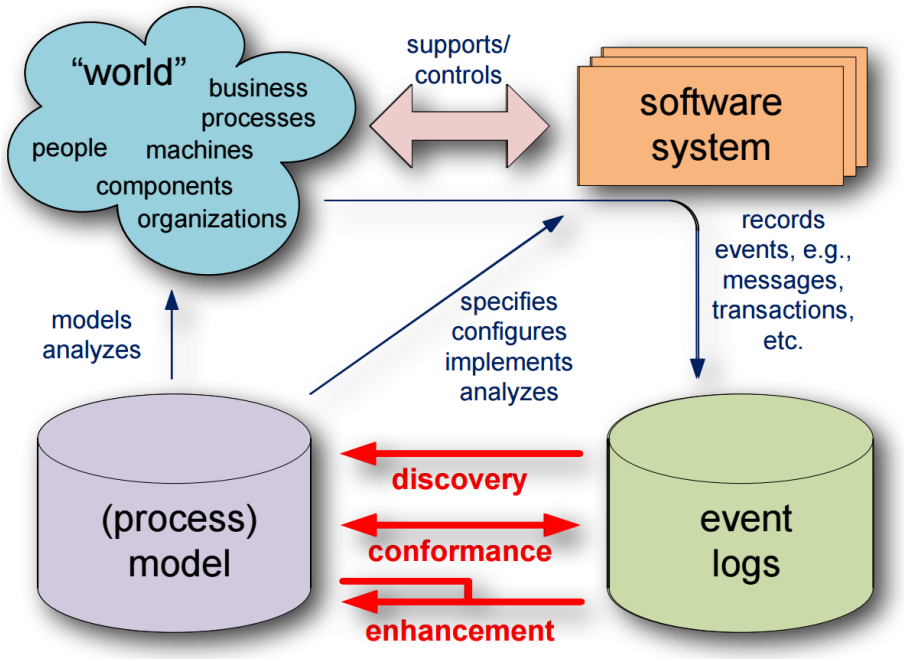


Fig. 2.8 The three main types of Process Mining (in red) [251]

level analyses (with human interaction) to support the diagnosis and design phase of the traditional BPM lifecycle, e.g. process discovery, what-if analyses, optimisation, etc. In a broader context Process Intelligence is regarded as a specialisation of *Business Intelligence* [44, 251].

In Process Intelligence two different types of analyses can be distinguished: Descriptive and Design-time Analyses. The former type encompasses all Process Mining (PM) techniques. The goal of Process Mining is to discover, monitor, or improve processes by harnessing information from an event log resulting from an actual BP execution, i.e. PM techniques essentially represent the link between event logs and business process models and enable an log-to-model analysis of the as-is state. As such, process mining is a research discipline located at the intersections between machine learning and data mining and between process modelling and process analysis [233]. Three main disciplines of process mining exist (see Figure 2.8) [251]:

1. *Conformance Checking,*
   i.e. determining to what extent the behaviour recorded in the log conforms to the behaviour of a given BP model and vice-versa

2. *Model Enhancement,*
   i.e. extending/annotating an existing model with additional information, e.g. performance data, obtained from the event log of this process, and

3. *Process Discovery,*
   i.e. extracting business process models from an event log without using any a-priori information [233]. Generally, process discovery is an umbrella term comprising the discovery of all perspectives of a business process, however, in most cases it is in fact

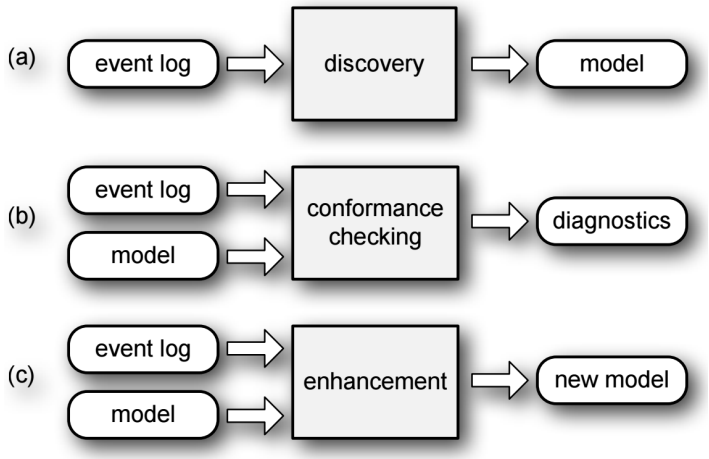


Fig. 2.9 Input and Output of (a) *Process Discovery*, (b) *Conformance Checking*, and (c) *Model Enhancement* [251]

only referring to the discovery of the control-flow perspective, e.g. [126, 249, 278]. Since most BPMS support the recording of an event log, this information can be readily accessed for analysis. However, the information in an event log is on a different abstraction level than a BP model. For this reason the extraction of BP-domain information from logs is a non-trivial task and has lead to extensive research in this area. The advances and state-of-the-art in process discovery will be discussed in detail in Section 2.4.

Figure 2.9 shows the different inputs and outputs of these three Process Mining disciplines, emphasising they essentially represent the link between execution event log and business process model.

The second type of analyses in Process Intelligence can be summarised as higher-level analyses that support the (re-)design stage. Those are mostly of a predictive or evaluative nature on either log-extracted or modelled BP data, or a combination of both. Examples are *verification* [259], *bottleneck detection* [195], *validation/what-if analysis* [73], *sensitivity analysis* [69], *optimisations* [221], or *business impact analysis* [189]. Methods like optimisation, sensitivity or what-if analyses are based on performance evaluations that employ methodologies for predicting future behaviour. This type of performance-related decision support is of high importance for reasoning and decision making and is discussed in further detail in Section 2.5.

The distinction between descriptive and design-time analyses is motivated to give an overview of the different types of decision support analyses in relation to the BPM lifecycle. The research areas and methods, e.g. performance prediction and process mining, do not have to be exclusively part of only one of the stages (diagnosis vs. analysis) but are assigned to either based on their prevalent usage. For instance, process mining techniques are predominantly used in the diagnosis phase but may also be used to some extent in the analysis phase (e.g. to compare actual and planned behaviour) or performance prediction on the mined model may also be part of the diagnosis phase.

Business Process Analysis in Real-time

The phase of the traditional BPM lifecycle (centre and right circle in Figure 2.7) are in many cases human-guided, i.e. not fully autonomic. However, making the parts of the lifecycle more automated is one of the general goals of BPM [214] and lead to extended research and a increasing number of publications in the area of automated business process analyses, e.g. [68, 106, 130], or management, e.g. [221, 268, 270]. These automated real-time analyses (and adjustments) differ from the static diagnosis and analysis methodologies, e.g. process mining and what-if analysis, that are prevalent in the traditional, human-guided BPM lifecycle. In order to account for the efforts in the area of automated BPA and BPM techniques the traditional lifecycle was extended in [251] with an additional cycle that represents these automated techniques that support analysis during BP execution/enactment (left cycle in Figure 2.7).

An alternative to this approach of log-based business process analysis is the immediate processing of events when they occur to information of an higher abstraction level in order to enable BPA in real-time. This is achieved with the help of *Complex Event Processing* (CEP), which is a method that essentially deals with the event-driven behaviour of large, distributed enterprise systems [60, 134]. This means in particular that events produced by the systems are captured, filtered, aggregated, and subsequently abstracted to generate complex events representing high-level information about the situational status of the system, e.g. current performance. By applying CEP methodologies in order to support a continuous analysis and management of business processes, Ammon et al. coined the term *Event-Driven Business Process Management* (EDBPM) [268–270]. The term emerged from the combination of the two disciplines Business Process Management and Complex Event Processing [270]. This is practically realised by two individual platforms interacting with each other through interfaces or events: One is a BPM system, which is used to model, manage, and optimise a business; the other one is a CEP engine [268].

Business Activity Monitoring (BAM) is often related to EDBPM as the real-time or near real-time approach of monitoring of events with a CEP engine to support BPM. In particular, BAM *"...refers to methods for the timely identification of threads and opportunities of business processes"* [179]. BAM is a term coined by the Gartner group and is mainly associated with monitoring a business's and its processes' performance in real-time by processing events from different business process execution sources [55]. The main goals of BAM is to allow immediate operational decision making based on real-time information [68, 292]. Single live-events are not of interest in the context of BAM, instead the aggregation of these into qualitative statements or quantitative performance measures is carried out [60, 68]. In Section 2.5 the state of the art for computing and predicting performance measures are discussed in detail.

The difference between BAM and EDBPM solutions is not clearly established and real-time monitoring solutions belong to both categories. Based on the definitions, it can be

argued that EDBPM may include automated higher-level analyses and small adjustments while BAM specialises on descriptive/diagnosis methodologies, i.e. BAM is a subset of EDBPM. At the moment a shift towards a process diagnosis at run-time is noticeable, reflected in an increasing number of publications detailing approaches about how to make modelling of BAM or EDBPM more automated, i.e. part of the business process modelling, e.g. [50, 68, 130, 149, 282]. Examples for types of current BAM or EDBPM solutions range from *process violation monitoring* [277] and *performance monitoring* [68, 107] over *performance prediction* [220] and *path prediction* [32] to initial work on automated *adjustment* [221, 270]. Note, that all of the referred BAM and EDBPM solutions in this section can be classified as model enhancement in the context of process mining terminology, since initial information about the process must already be provided, i.e. modelled.

2.3 Flexibility in Business Processes

Business processes are required to adapt to dynamic changes in the environment [203], e.g. because of a lack of available resources, introduction of a new quality assurance step, or simply due to an optimisation of the existing control-flow. For instance, in domains like health care, Customer Relationship Management (CRM), or customised product manufacturing, process adaptations are necessary or desirable to address changing demands. The ability to adapt to change without losing one's core identity is called *flexibility* [190]. For descriptive run-time models of business processes it is important to reflect change and thus to analyse which types of changes a business process can be subject to. For this reason this section provides a state-of-the-art discussion on change-induced flexibility in business processes.

2.3.1 Taxonomy of Business Process Flexibility

Research in the field of business process flexibility has resulted in a number of publications with the goal to classify different types of business process flexibility, e.g. [190, 192, 202, 211, 238]. A generic taxonomy of business process flexibility is proposed by Regev et al. in [190]. The presented taxonomy is composed of three dimensions of change criteria (see Figure 2.10):

1. *Abstraction Level of Change*: As discussed earlier (Section 2.1) two abstraction levels are commonly distinguished in BP terminology: (1) process type, which describes the general process and its business goals, and (2) process instance, which is a single execution of the process. Changes on the process instance level denote deviation from the current definition of the process type and are usually of exceptional nature. On the other hand changes on the process type level constitute a shift/evolution of the process definition which affects all future (and possibly current) instances.

2. *Subject of Change*: A business process change can affect one or more perspectives of

business process, e.g. the control-flow, the resources, and/or functional perspective (see Section 2.1).

3. *Properties of Change*: Four different properties of change are considered: (1) extent, i.e. whether the change is of incremental or revolutionary nature, (2) duration, i.e. whether the change is temporary or permanent, (3) swiftness, i.e. whether the implementation is immediate or deferred, and (4) anticipation, i.e. whether the change is planned or ad hoc.

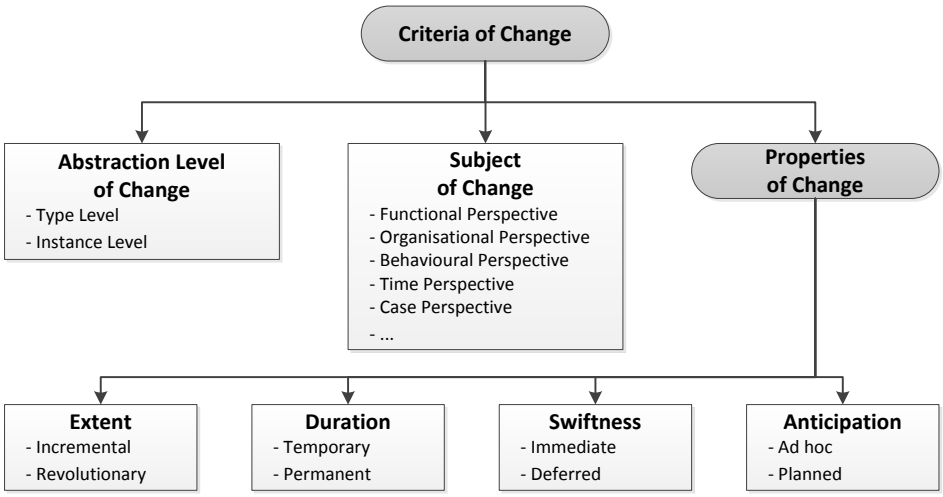


Fig. 2.10 Taxonomy of Business Process Flexibility by Regev [190]

Note, that some of the criteria are inherently connected, e.g. a change in the case/data perspective is always a change on the instance level, or mutually exclusive, e.g. any change on the instance level can never be of a revolutionary extent. A similar taxonomy based on change criteria is proposed in [238] with less focus on instance level changes but additional criteria defining what happens with currently active instances in case of a change on the process type level, i.e. modification policies (further discussed in Section 2.3.2, Run-time Flexibility).

Another, taxonomy defining five different types of process flexibility which were derived from flexibility mechanisms supported by current BP languages is presented by Schonenberg et al. in [211]:

1. *Flexibility by design* is the ability to model alternative execution paths within the process definition at design-time. Dependent on the circumstances, the most appropriate execution path for a process instance can be chosen at run-time. This dimension is supported by any business process modelling language to some extent.

2. *Flexibility by deviation* is the ability for a process instance to deviate at run-time from the prescribed execution path of the business process model. The deviation does not allow for changes in the process definition, i.e. the business process model.

3. *Flexibility by underspecification* is the ability to execute an only partially defined business process at run-time. The full specification of the model is made at run-time and can be unique for each process instance.

4. *Flexibility by momentary change* is the ability to modify the execution of one or more selected process instances. This change is performed at the process instance level and does not affect any future instances.

5. *Flexibility by permanent change* is the ability to modify business process model at run-time such that the process definition is permanently modified. All currently executing process instances need to be transferred to the new process definition.

Another similar differentiation can be found in [204], in which dimensions of change for workflows are defined. Note, that the terminology in the following approach is a little contradictory to the terminology used in the first approach, i.e. "flexibility" does have a slightly different meaning and is therefore notated with a star ("flexibility*"). The classification of change dimensions for workflows is [133, 202, 204]:

1. *Flexibility** is the ability of the workflow process to execute on the basis of an incompletely specified model, where the full specification of the model is made at runtime [204]. This dimension of change is the equivalent of *flexibility by underspecification* of the previous taxonomy. Arguably *flexibility by design* can also be affiliated with the flexibility* dimension of Sadiq et al.: Designed flexibility, e.g. via a decision or fork, could be understood as an incompletely designed process only fully specified for each instance at run-time.

2. *Adaptability* is *"... the ability of the workflow processes to react to exceptional circumstances"* [204]. These exceptional circumstances would affect one or more instances but not the underlying business process on the type level. This dimension of change is comparable to *flexibility by momentary change* or *flexibility by deviation* of the previous taxonomy dependent on if the process definition is momentarily adapted or not.

3. *Dynamism* is *"... the ability of the workflow process to change when the business process evolves. This evolution may be slight as for process improvements, or drastic as for process innovation or process reengineering"* [204]. Compared to the previous taxonomy this dimension is equivalent to *flexibility by permanent change*.

2.3.2 Build-time vs. Run-time Business Process Flexibility

Generally flexibility in processes are either anticipated or the result of an intervening action. The taxonomies of Sadiq et al. and Schonenberg et al. allow a clearer differentiation between two types of flexibility mechanisms:

- *build-time flexibility*, i.e. the ability to pre-model flexible execution behaviour, and

- *run-time flexibility*, i.e. in which an adaptation at run-time in the sense of exception handling or process evolution is carried out.

In both cases the challenge is to balance flexibility and control [202]. Figure 2.11 shows how the flexibility types of Sadiq et al. (light grey rectangles) and Schonenberg et al. (dark grey ovals) relate to level of abstraction (type vs. instance) and anticipation (build-time vs. run-time).

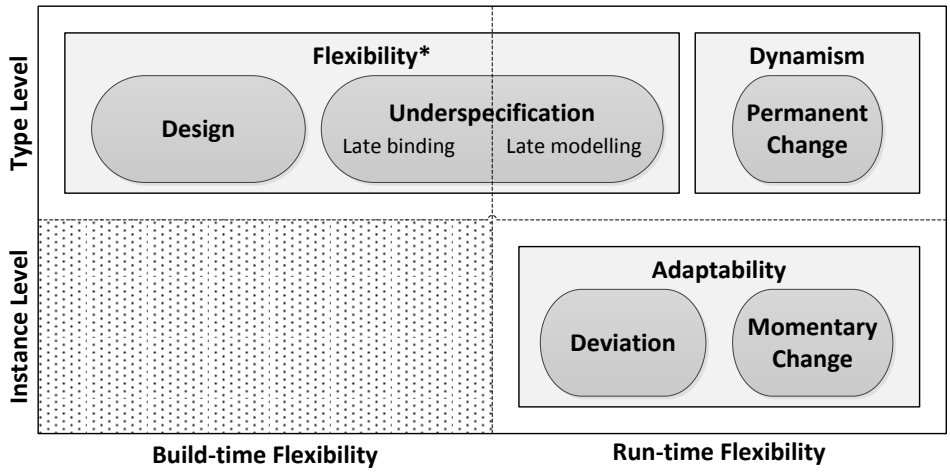


Fig. 2.11 Types of BP Flexibility defined by Sadiq et al. [202] (light grey rectangles) and by Schonenberg et al. [211] (dark grey ovals) in relation to level of abstraction (type vs. instance) and anticipation (build-time vs. run-time)

Build-time Flexibility

Build-time flexibility is about leaving parts of the business process unspecified at design-time, i.e. the flexibility is modelled into the business process, and the missing information is determined at run-time according to pre-specified constraints or rules. Different approaches to achieve this type of flexibility are by applying either general declarative processes [171, 240, 281], advanced modelling [237], or late-binding [202]. In contrast to the traditional imperative business process models (see Section 2.1) declarative business processes are models that focus on the specification of what is not allowed instead of what is allowed and are therefore generally more suitable to support flexibility characteristics [133]. Declarative processes are particularly popular in academia but have not yet achieved a high acceptance in industry. On the other hand, flexibility through advanced imperative business modelling became to a certain extent a prevalent means to incorporate flexibility in a business process, e.g. the synchronising fork-join behaviour of the example in Figure 2.2 allows for a flexible execution order of the activities on the respective parallel paths. Pioneers of the last approach of late-binding are Sadiq et al. who introduced so called *pockets of flexibility* for workflow specifications [202]. The introduced workflow specification consists of [204]:

- core process consisting of pre-defined activities,
- pockets of flexibility within the process which in turn consist of
  - set of process elements, which can be a single activity or a sub-process,
  - set of constraints for concretising the pocket with a valid composition of process elements.

The definition is recursive and thus supports a hierarchical declaration of flexibility pockets.

Run-time Flexibility

Since business process designers are not capable of anticipating all possible cases, exceptions, and events beforehand, the run-time system may not have sufficient knowledge to handle these situations. The second type of flexibility is about permanently or temporarily adapting the business process model or deviating from it at run-time, i.e. run-time flexibility. Temporary adaptations in the sense of ad-hoc changes are carried out on the process instance level and supported by exception- or case-handling [250], thus corresponding to the *Adaptability* dimension (see Figure 2.11). Permanent adaptation in the sense of process evolution is carried out by process schema changes on the process type level (see *Dynamism* in Figure 2.11) and, for instance, supported by adaptive workflow languages like ADEPT$_{flex}$ [191] or A$_{GENT}$W$_{ORK}$ [154].

[273**?**]

One challenge in the field of run-time flexibility is the *validation* of the change that is to be applied to the system. In [191] a conceptual and operational framework is proposed that can reason about the correctness of a requested change to handle dynamic structural adaptations of workflows. At the core of this framework is a conceptual graphical workflow model (ADEPT) based upon which a complete and minimal set of change operations (ADEPT$_{flex}$) is defined, e.g. dynamic insertions/deletion of activities, or changing activity sequence. These operations allow for modifying the structure while preserving correctness and consistency of the system. With the help of formal constraints for state, flow of data, and flow of control, changes can be rejected if they can potentially lead to an invalid state of the system [192]. This solution provides only a minimal set of changes with strict constraints to ensure that no invalid state can be reached. A more coarse-grained view on these changes can help to reduce or relax these constraints, i.e. grouping changes instead of regarding every change as an atomic modification action. For instance, assuming two previously sequential activities are to become parallel, the constraints for this coarse-grained modification would be less strong than the constraints of the sub-modifications, deletion and parallel insertion, regarded individually. Weber et al. [272, 273] identified 18 change patterns based on 157 real-life business processes from the domains of health care and automotive. 14 of these are adaptation patterns of different granularity, e.g. insert

process fragment, delete process fragment, swap process fragments, parallelise activities, and embed process fragment in loop. The identified changes only consider the control-flow perspective and would have to be extended by patterns for the other perspectives, e.g. reallocation of resources.

If more complex changes, e.g. to parallelise activities, are requested *modification policies* (also called *migration strategies* in some literature) have to be in place to ensure that the run-time system continues to operate in the expected manner. Modification policies specify how the transition from one business process to another is carried out [203]. These policies are important with respect to the still active process instances of the outdated business process and describe how to deal with them. Example policies are *Flush*, which allows all current instances to complete according to the old process model, *Abort*, which aborts all active process instances, and *Migrate*, which maps the state of active process instances to the new model. The last option is only applicable if additional migration constraints can be met, i.e. the migration into a valid instance is possible. Modification policies are discussed in more detail by Sadiq et al. [203] and Schonenberg et al. [211].

Despite the research effort carried out in the field of run-time flexibility of workflows (and BPs) the resulting solutions have not yet found a wide acceptance in industry (mostly only in the area of fully automated workflows). This can be mainly attributed to the complexity and often perceived overhead of addressing the challenges of validity and change policies. If a framework does not support run-time flexibility in the sense of dynamism (changing BP model at run-time) changes are often introduced by *"... tossing aside existing processes and starting over"* [247], i.e. carrying out a hard replacement of business process definitions or systems. This approach conforms more to the view of the BPM lifecycle shown in Figure 2.3 which advocates separate diagnosis and (re-)design steps carried out offline rather than online before the new model is enacted. However, some BPMS like ADONIS[5] allow to import new workflow models during run-time as shown by Herbst et al. in [96]. Another source for run-time flexibility are business processes definitions that are not fully enforced by the system (e.g. for documented, manual, or semi-automatic BPs), and thus already allow for exceptions, deviations, and evolution within a certain margin.

2.4 Process Discovery

The research area of *Process Discovery* is concerned with the extraction of a business process model from event logs without using any a-priori information [233]. Generally, process discovery is an umbrella term enclosing the discovery of all perspectives of a business process, however, in reality solutions published in the topics of process mining or process discovery are mostly concerned with the discovery of the control-flow perspective since it poses the most difficult of the challenges.

[5] ADONIS is a registered trademark by BOC GmbH

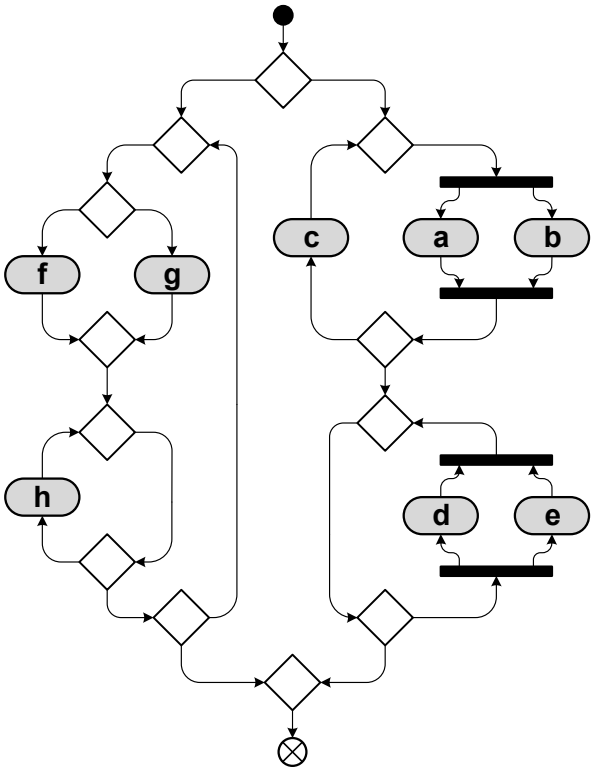


Fig. 2.12 Example Business Process

2.4.1 The Process Discovery Problem

Traditional process discovery approaches are carried out in a static way as an "offline" method contrary to event-based BAM or EDBPM solutions (see Section 2.2.3). This is reflected by the fact that the input for these algorithms is an entire event log. Different to the generic event logs introduced in Section 2.2.2, it is sufficient for the discovery of a control-flow to contain only a minimal set of event features: Every event needs to have a reference (1) to its *process instance*, e.g. via unique identifier, and (2) to the corresponding *activity*, e.g. via unique name. Furthermore, it is assumed that the log contains exactly one event for each activity execution, i.e. activity lifecycle events are not regarded (e.g. start, assign, complete). In the context of process discovery, the execution of any two process instances is assumed to be independent, i.e. the execution order within an instance does not depend on the execution order of a second instance. In terms of process log terminology (see Section 2.2.2), all events resulting from the execution of the same *process instance* are captured in one *trace*, which is sometimes also called *case*. Accordingly, an event is represented by a pair (t,s) where t links to the trace and s to the activity. Figure 2.12 shows an example process consisting of simple one-letter activities: a,b,c,d,e,f,g,h. This example BP represents the reference process to help explaining the log-model terminology and concepts. Since two traces are assumed to be independent from each other only the order of the activities within a trace is of interest, i.e. a trace can be specified by a sequence of activities ordered by their occurrence: $t_1 = [a,b,d,e]$ denotes a trace that conforms to the process in Figure 2.12.

Furthermore, traces only consisting of the activity order are called *simple traces* and event logs only consisting of simple traces are called *simple event logs* [233]. An example of a simple event log for the business process in Figure 2.12 is [6]

$$L_1 = \{[b,a]^4, [a,b,d,e]^5, [b,a,e,d]^4, [b,a,c,a,b,c,b,a,d,e,e,d]^6, \\ [g,g]^2, [f,h]^3, [f,f,h,f,g,h,g,f,h]^8, [g,h,f]^2\} \tag{2.1}$$

The log L_1 consists of eight different traces each occurring a number of times. The position of the traces in the log is of no relevance for process discovery algorithms since all have the same "weight" and need to be taken into account equally. Note, that log L_1 does not include all possible behaviour of the process from Figure 2.12, e.g. even though in the model activity b may be followed by activity e no trace in L_1 contains this behaviour. Logs that do not include all possible behaviour of a source model are called *incomplete logs* [128] and emerged as a challenge for rediscovering BPs. Another challenge originating from real life problems is the analysis of logs that contain exceptional/infrequent behaviour. Behaviour that does not conform to the source model is called *noise* or *infrequent behaviour* [127]. For instance, adding trace $[a,b,d,e,e]$ to log L_1 would result in an incomplete event log containing infrequent behaviour since the new trace is not conforming to the source model, i.e.

$$L_2 = \{[b,a]^4, [a,b,d,e]^5, [b,a,e,d]^4, [b,a,c,a,b,c,b,a,d,e,e,d]^6, \\ [g,g]^2, [f,h]^3, [f,f,h,f,g,h,g,f,h]^8, [g,h,f]^2, [a,b,d,e,e]^1\} \tag{2.2}$$

Considering that real-life logs are incomplete and/or contain infrequent behaviour, conventional challenges in process discovery originate from the motivation to find the "best fitting" model, i.e. discovered processes should be an accurate reflection of the behaviour contained in the log. Determining the "best fitting" model that conforms to a given log is essentially a balance between *over-fitting* and *under-fitting* [46, 243]:

- An *over-fitting* model does allow for only the exact behaviour recorded in the log, i.e. it does not generalise any behaviour and therefore does not allow for any additional behaviour [254]. A naive example for an over-fitting model is the *Trace Model*: Here every possible different trace of a log is transferred into an activity sequence combined to a choice construct (see left model in Figure 2.13 for log L_1). The example model as well as trace models in general contain duplicate activities.

- An *under-fitting* model is *"overgeneralised"* [243] which is usually synonymous for allowing *"... too much behaviour that is not supported ..."* [254] by the corresponding log. Considering log L_1, a model that would allow any permutation the involved activities is considered under-fitting. A naive example would be the *Flower Model*

[6] The power values denote the respective occurrences of the traces in the log, e.g. trace $[b,a]$ occurs 4 times in the log.

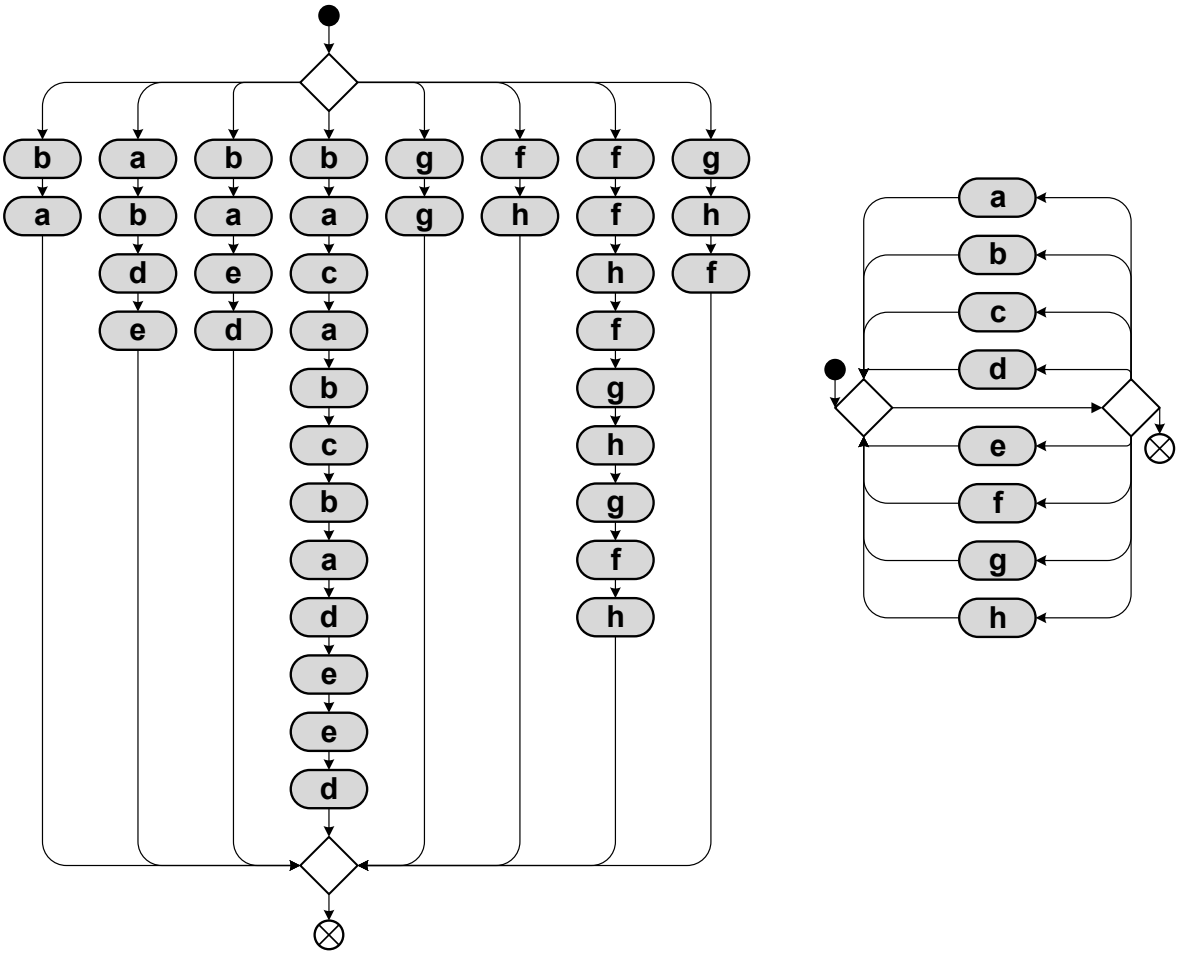


Fig. 2.13 Mined *Trace Model* (left) and *Flower Model* (right) for Log L_1

(see right model in Figure 2.13) which allows for all permutations (with infinite repetition) of a, b, c, d, e, f, g, h.

Both, Trace Model and Flower Model, are considered evaluation baselines for over-fitting and under-fitting models, respectively. The quality of process discovery approaches is essentially determined by their ability to discover the "best fitting" model from a given event log, i.e. evaluation based on log-model conformance checking techniques (see Figure 2.9). What constitutes a "best fitting" model strongly depends on given requirements and goals, e.g. the three BP models in Figures 2.12 and 2.13 are each a valid representative of log L_1 and have their respective advantages and disadvantages. Thus, process discovery represents a multi-goal optimisation problem, i.e. a trade-off between the following four quality criteria [233]:

1. *Fitness*: The ability of the discovered model to allow for the behaviour recorded in the event log [3] - in other conformance checking literature and general data mining terminology this quality criteria represents the *recall* measure [146].

2. *Precision*: The ability of the discovered model to not allow for behaviour unrelated to what is recorded in the event log [2, 146, 155].

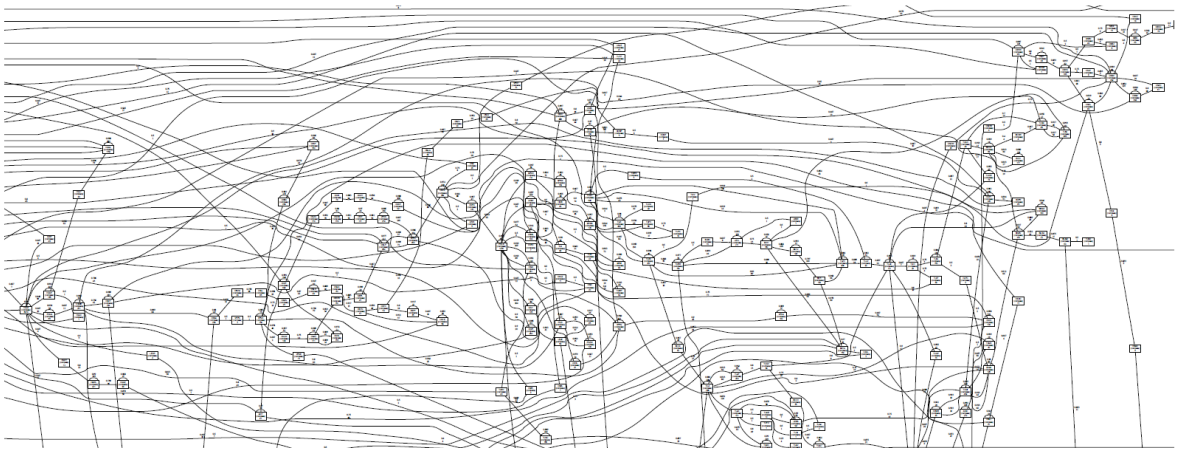


Fig. 2.14 Excerpt of a large "Spaghetti" Process Model [89]

3. *Generalisation*: The ability of the discovered model to have generalised the *example* behaviour recorded in the event log [236].

4. *Simplicity*: A measure to avoid overly complex models, e.g. a "spaghetti" model [89] as shown in Figure 2.14.

With respect to those four criteria under-fitting models like the flower model score well for fitness, generalisation, and simplicity values but very poorly for the precision value, and over-fitting models like the trace model score well for the fitness and precision value but poorly for generalisation and simplicity. The fitness and precision criteria originate from traditional data mining challenges and evaluate the accuracy of a mined model. On the other hand the criteria of generalisation and simplicity emphasises the need to create less complex and human-readable models. Disregarding these criteria in favour of the accuracy-based criteria often lead to the discovery of overly complex and highly interconnected business processes, so called "spaghetti" models (see Figure 2.14) [89, 234]. This is especially the case when dealing with logs that contain incomplete and/or infrequent (noisy) behaviour. In recent literature, e.g. in [26, 118, 126, 147, 148, 175, 234], an increasingly popular notion to promote generalisation and simplicity is the restriction of the BP model language to a structure that only allows for constructs which are easy to understand. The assumption here is that a well structured model is far easier to analyse while still being able to represent the main behaviour recorded in the log, i.e. a little accuracy (fitness and precision) is sacrificed in favour of a much increased structuredness [118].

A BP model is *"... (well-)structured, if for every node with multiple outgoing arcs (a split) there is a corresponding node with multiple incoming arcs (a join), and vice versa, such that the fragment of the model between the split and the join forms a single-entry-single-exit (SESE) process component"* [175]. The processes shown in Figure 2.2 and Figure 2.12 are examples of such well-structured processes. This definition allows to introduce a hierarchy of different constructs within a process, i.e. supports separations and drill-down capabilities that help to understand independent parts of the process individually. For instance, the example process in Figure 2.12 is essentially a choice between the

executions of activities a, b, c, d, e (right branch) or f, g, h (left branch); the right branch is essentially a sequence of activities a, b, c (top right) followed by activities d, e (bottom right), and so on. BP models conforming to this definition of (well-)structuredness are also called *block-structured* BP models in the context of this thesis. Other representations with a very similar structure definition are *Hierarchical Process Models*, an abstracted view on block-structured BPMN-like process models [147, 148], and *Process Trees*, a simplified notation of block-structured Petri Nets/workflow nets [26, 126].

2.4.2 Process Discovery Algorithms

The discovery of the trace and flower model in Figure 2.13 from log L_1 are trivial solutions for the process discovery problem. However, a large number of advanced process discovery algorithms exist, each with its own respective strengths and weaknesses. Pioneers of process discovery were Mannila et al. who proposed an algorithmic method for discovery of frequent episodes in event sequences to describe the behaviour of users or systems [140]. The first application to the workflow domain to discover directed graphs from worklfow logs was carried out by Agrawal et al. [4]. Cook and Wolf presented three approaches to the grammar inference problem in order to discover software processes from event data based on statistical and/or algorithmic methods such as Neural Nets and Markov Models [39].

As the WfMS or BPMS became more evolved and more event data in terms of quantity, quality, and diversity was available new challenging problems emerged, e.g. detecting hidden or duplicated activities, non-free choice constructs, and loops or dealing with incomplete and/or noisy logs [248]. As a result, in the last twenty years a large number of new process discovery approaches have been developed. A selection of the most prominent approaches are listed in the following categories:

Abstraction-based Approaches

These are process discovery algorithms that *"... construct a net based on an abstraction of the log"* [254]. The employed mining procedure consists of three phases [254]:

1. *Abstraction phase* in which each trace of the event log is analysed for direct successions of activity occurrences in order to create basic ordering relations, e.g. for log L_1 activity c is directly followed by a or b (see the fourth trace in Equation 2.1),

2. *Induction phase* in which advanced ordering relations are induced from the previously abstracted basic ordering relations, e.g. activities a and b have a "parallel" relation because in each case if a occurs it is followed by b and vice versa, and

3. *Construction phase* in which the BP model is constructed from the inferred relations of the induction phase.

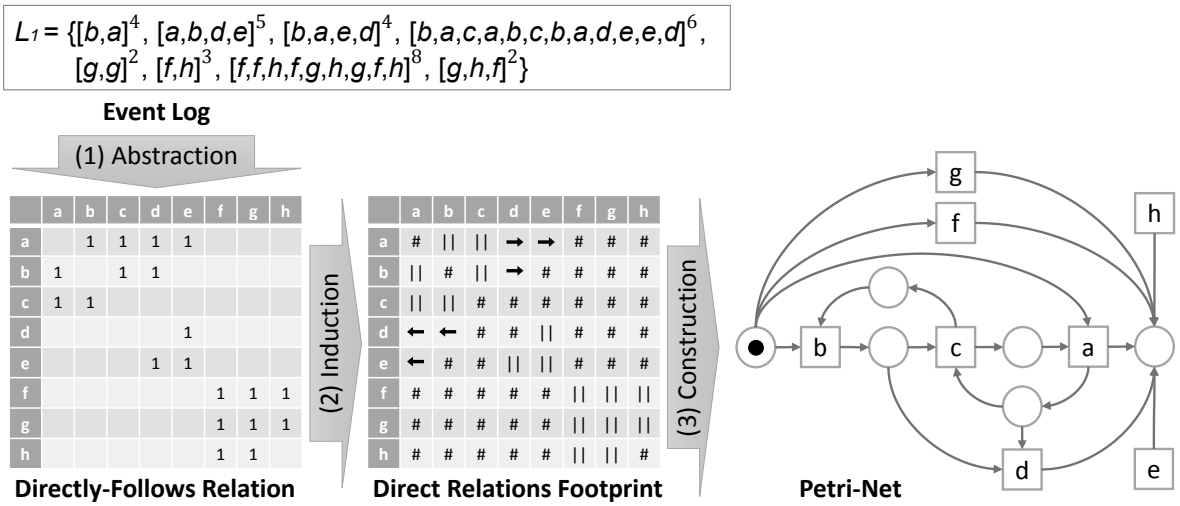


Fig. 2.15 Concept of the α-algorithm: Intermediate and End Results for Log L_1

The foundation approach of abstraction-based algorithms is the *α-algorithm* [249] which discovers a BP model in form of a Petri Net [172] from a given event log. Figure 2.15 illustrates the general concept of the algorithm for the example log L_1 (Equation 2.1): First, in the abstraction phase all directly-follows relations occurring at least once in the log are capture, e.g. trace $[a,b,d,e]$ contains the relations $a > b$ (b directly follows a), $b > d$ (d directly follows b), and $d > e$ (e directly follows d), i.e. $> = \{(a,b),(b,d),(d,e)\}$. The left table in Figure 2.15 shows all directly-follows relations contained in log L_1 with "1" marking contained pairs. Then, in the induction phase the directly-follows abstraction is processed to three exclusive footprint relations: (1) $x\#y$ if neither $x > y$ nor $y > x$, (2) $x||y$ if both $x > y$ and $y > x$, and (3) $x \rightarrow y$ if $x > y$ but not $y > x$. The respective relations between any activities $x, y \in \{a,b,c,d,e,f,g,h\}$ are shown in the middle table of Figure 2.15. This is followed by the construction phase in which typical local process patterns are identified from the footprint, e.g. if $a \rightarrow d$, $b \rightarrow d$, and $a||b$, then a and b are the parallel inputs for the transition representing d (see bottom of Petri Net in Figure 2.15: transitions a and b have to be triggered before d can be triggered). The Petri Net shown is the final result of the α-algorithm for log L_1 discovered with the help of the *ProM* [7] framework [253]. Note, that due to problems discussed in the next paragraph the Petri Net discovered by the α-algorithm has a lot of flaws and is not usable, e.g. activity transition b (and thus also a, c, and d) can never fire because a token in the second input place is missing. Additionally, a number of variations or extensions of the α-algorithm have been published which either enhanced the abstraction phase, e.g. dealing with non-atomic tasks [280], or the number of detectable constructs, e.g. non-free choice constructs [279] or short loops [45].

Abstraction algorithms based on the α-algorithm are accuracy-driven, factual analyses using the directly-follows abstraction to discover BP models in the form of a Petri Net (a General Purpose Language). As such, they discover models that try to represent *exactly* the behaviour contained in the log, irrespective of completeness or occurrence frequency

[7] ProM is an academic framework that provides a set of tools and plugin implementations of published algorithms related to the discovery and analysis of business processes (see http://www.processmining.org).

of certain behaviour: a relation either exists in the log or it does not. As a result, analysing logs containing incomplete or infrequent (noisy) behaviour with these algorithms will often lead to the discovery of poorly generalised BP models, i.e. either unsound (e.g. see result in Figure 2.15) or overly complex (e.g. "spaghetti" models like shown in Figure 2.14) BP models. In recent literature two general notions evolved to avoid the discovery of poorly generalised BP models: Firstly, to promote the discovery of well-structured processes (see Section 2.4.1) and, secondly, to consider the frequency of occurring behaviour (discussed in the next Section "Heuristic-based Approaches").

With regards to the first notion of discovering only well-structured (i.e. block-structured) processes from a log, one option is the discovery of an arbitrary Petri Net followed by a transformation into a block-structured process as shown in [175]. While this approach improves the human-readability through the introduction of a clear hierarchy, its application to unsound Petri Nets like the one in Figure 2.15 is not possible. Another option is the direct discovery of block-structured BP models. An algorithm that directly discovers block-structured Petri Nets is the *Inductive Miner* (IM) [126]. Like the α-algorithm the IM is based on the directly-follows abstraction of an event log but instead of inducing local relations, i.e. identifying splits, joins, decisions, etc., it induces single-entry-single-exit constructs, i.e. loop, parallelism, etc. For this the IM evaluates constraints based on the existing directly-follows relations to identify the construct that represents the specified set of activities. This is done following a divide-and-conquer approach from top till down. That means, first the parent construct is identified which splits the set of activities into a number of subsets, then these subsets are analysed the same way to identify their respective constructs, and so on. The result of the induction phase is a process tree from which then the Petri Net is constructed. The original IM focuses on rediscovering models from complete logs without noise. However, if an incomplete and/or noisy log is analysed it still guarantees to find a sound model while favouring fitness over precision[8]. In order to improve the results of the original IM two more extensions have been proposed: In [127] special filters are introduced to better deal with noise and in [128] probabilistic behavioural relations are introduced to improve the discovery results for incomplete logs. Figure 2.16 shows the result of the improved Inductive Miner for log L_1 extracted with the IM plugin readily available for the ProM framework. It is a sound Petri Net model with an (semi-)hierarchical structure that represents the main (but not all) behaviour present in the log. Note, that possibly due to optimisation efforts in the transformation from process tree to Petri Net the hierarchical structure was not entirely maintained: After activity c it is possible to transition into the part of the net with activities f, g, h. This is not allowed in the original process tree discovered by the IM since it supports an exclusive decision between activities a, b, c, d, e and activities f, g, h.

[8] If the conditions for none of the other construct is satisfied, a construct similar to the flower model is build, thus producing a solution that allows for all behaviour with leads to a low degree of precision

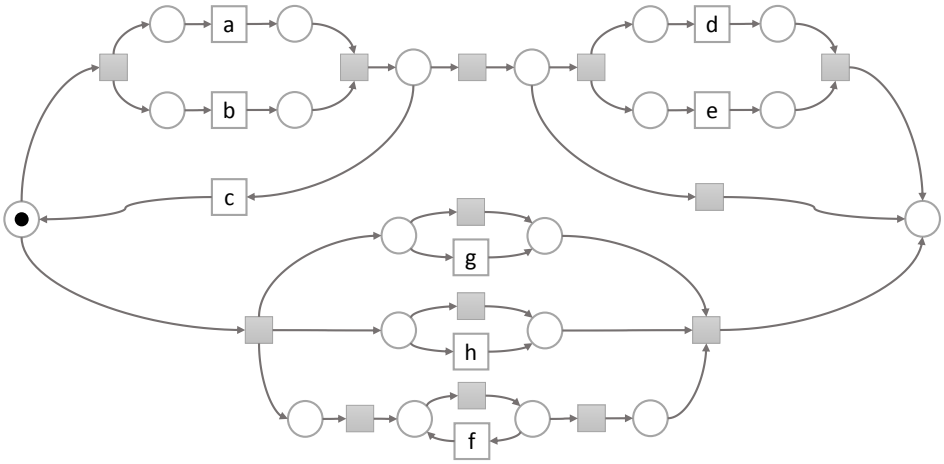


Fig. 2.16 Result Petri Net of the Inductive Miner Plugin in ProM for Log L_1

Heuristic-based Approaches

The abstraction-based algorithms and extensions discussed in the previous subsection are designed to perform especially well for rediscovering business processes, i.e. if the log contains exactly the behaviour of a source BP model. This, however, is seldom the case in real-life scenarios in which logs often contain exceptional behaviour, i.e. noise. One simple technique to avoid this is to pre-filter the log and remove non-frequent traces. This technique may improve the readability of a general purpose target language like Petri Net but is not always applicable for more restricted languages like process trees since it could accidentally remove behaviour important for the identification of advanced ordering relations, i.e. the post-filtered log becomes incomplete. Many recent algorithms therefore take the occurrence frequency of behaviour in the log directly into consideration when discovering a BP model, using so called *heuristics* for the abstraction of the log content. For instance, the extension of the Inductive Miner to deal with noise uses a heuristics-based filter among other techniques [127]. Another example is the *Fuzzy Miner* [89] which clusters events using correlations and significance measures to reduce the "exact" BP model to a simplified model that still supports the main behaviour of the process. While the resulting "fuzzy model" is particularly well suited for getting an initial understanding of an unknown process, it does not conform to the usual workflow format of the BP-domain and has no executable semantic which would allow for the transformation to a BP-domain language or analyses of a higher level, e.g. optimisation, conformance, simulation.

A very prominent process discovery algorithm is the *HeuristicsMiner* (HM) [278] which mines a *Causal Net*, a representation tailored towards Process Mining [233]. Figure 2.17 shows the concept and result of the HM for the example log L_1 [278]: Firstly, instead of the directly-follows relation $x > y$ for $x, y \in \{a, b, c, d, e, f, g, h\}$ the number of occurrences of these relations in the log is considered, i.e. $|x > y|$ (left table in Figure 2.17). With the help of the directly-follows relation count the respective dependency measures $x >> y$ are

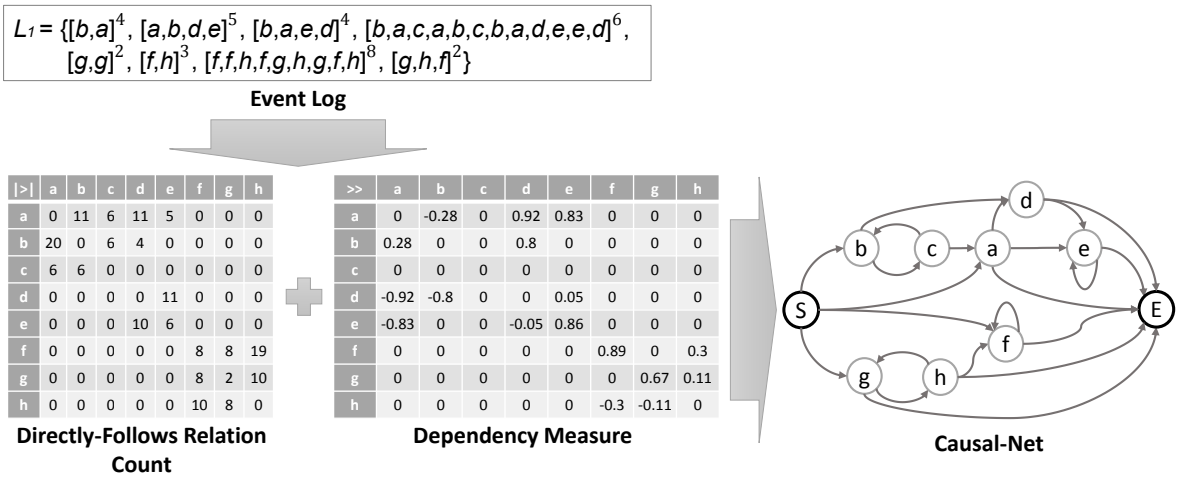


Fig. 2.17 Concept of the HeuristicsMiner: Intermediate and End Results for Log L_1

calculated which represent the footprint (centre table in Figure 2.17):

$$x >> y = \begin{cases} \frac{|x>x|}{|x>x|+1} & \text{if} \quad x = y \\ \frac{|x>y|-|y>x|}{|x>y|+|y>x|+1} & \text{if} \quad x \neq y \end{cases}$$

Note, that the footprint used in the HeuristicsMiner is not based on absolute relations between activities but on relative relation values. In a third step, the directly-follows counts and dependency measures are processed to a Causal Net, depending on pre-specified thresholds for both measures. By increasing or decreasing these thresholds the resulting Causal Net may include less or more of the causal connections between activities. Special treatment not shown in the concept in Figure 2.17 is applied for short loops (e.g. $\{a,b,a,b,a\}$) and parallel splits or joins. The resulting Causal Net for log L_1 on the right in Figure 2.17 represents the main behaviour in the log according to the interpretation of the HM. However, some connections are neglected, possibly due to their relative infrequent occurrence in the log, e.g. c can only be executed after b but not after a. Plugins for the transformation of the resulting Causal Net to a more common process representation, e.g. BPMN model or Petri Net, are available in the ProM framework.

Other Related Approaches

Additionally to the abstraction-based and heuristic-based algorithms discussed previously a number of other techniques exist to deal with the process discovery problem. Other than employing fuzzy models [89] or neural networks [39], the usage of genetic/evolutionary algorithms are an emerging third type of artificial intelligence techniques that enjoys an increased popularity among process discovery algorithms, e.g. [26, 46, 147, 148, 229, 257]. These algorithms adapt evolutionary principles like mutation, recombination, and selection to explore the solution space in order to converge a population closer to the optimum with each new generation. The involved operators mutation, recombination and sometimes selection are, like in nature, randomly influenced which makes ge-

netic algorithms a non-determinstic approach. Generally, the application of such evolutionary, search-based algorithms has two benefits: One is that it is independent from local information in the log, i.e. instead of directly inferring from local relationships, e.g. directly-follows relation, only the overall conformance measure is relevant with regards to the "fitness" of an individual. This technically allows for better results as the conformance measure that eventually defines the quality of the discovered model is directly involved in the algorithm. A second benefit is that the target language can be freely chosen and optimised according to the specified conformance measure. As a result, the best fitting block-structured BP model can theoretically be discovered despite incomplete or noisy logs as shown in [26] and [147]. The apparent disadvantages are (1) the non-deterministic nature of the genetic algorithms, i.e. despite the same input log every analysis run may yield a new result, and (2) the potentially very high exponential run-time, i.e. since the search space is expanding exponentially it cannot be guaranteed that a well fitting model is discovered in finite time (especially for large models). One approach to minimise the time consuming conformance and quality measures is to replace them by a quicker approximate conformance measure which yields similar results as shown in [146–148]. This allows for a quicker convergence in exchange for a little less accuracy but essentially does not eliminate the two general main disadvantages of genetic algorithms: their exponentially rising run-time and non-deterministic nature.

Yet another type of process discovery techniques is based on the *Theory of Regions* which is concerned with the discovery of a Petri Net from a behavioural specification in order to synthesise behaviour that has not yet been observed, i.e. incomplete logs [243]. The goal is to construct a Petri Net "... such that the behavior of this net corresponds with the specified behavior, if such a net exists" [254]. In [243] a step-wise methodology using *state-based regions* to construct a Petri Net from a transitions system (which has previously been constructed by connecting and merging each individual "state" of each trace/instance in the log) is proposed. Unfortunately, these transition systems may become excessively big considering that loops and parallel behaviour lead to possibly infinite states (for loops) or exponentially growing number of states (for parallel splits with many activities). This was addressed by limiting the definition of a trace's "state" to the last n events which has the negative effect of potentially neglecting behaviour needed to identify loops or splits spanning sequences longer than n. This approach is therefore strongly dependent on manual effort to find the right balance between "under-fitting" and "over-fitting". Instead of state-based regions other approaches promote the extraction of Petri Nets with the help of *language-base* regions, i.e. a net synthesis based on a language specification, e.g. [13, 132]. Regarding a business process as a language where each event is a letter in an alphabet and each trace a word of the language has its own shortcomings, e.g. no "invisible" transitions and difficulties to express "parallelism". Their detailed discussion is, however, not in the scope of this thesis. Generally, algorithms based on the theory of regions are factual analyses (like the α-algorithm) focussing on the synthesis of behaviour not present in the log but do not address the problem of infrequent

behaviour, i.e. noise.

2.4.3 Concept Drift in Process Mining

Concept drift is a problem in the domain of online machine learning that describes changes in a *hidden context* that potentially cause more or less radical changes in the target concept [284]. That is, changes in the underlying data often make the model built on the "old" data inconsistent and can only be addressed with regular model updates to reflect the new circumstances. The problem of concept drift complicates the task of learning a model and requires a deviation from common techniques which treat every instance (old and new) with equal importance [228].

The same applies in the domain of process discovery: Models of business processes may change over time (see Business Process Flexibility Section 2.3), i.e. an event log may not only contain information of one BP but in fact of multiple versions/variants of a process. If we take the example log L_1 (see Equation 2.1 on page 35) and assume that all traces in the first line occurred before all traces of the second line then it is fair to assume that the execution of two (completely) different versions of the process was recorded in one log. If L_1 is split into two separate logs

$$L_1^* = \{[b,a]^4, [a,b,d,e]^5, [b,a,e,d]^4, [b,a,c,a,b,c,b,a,d,e,e,d]^6\} \text{ and}$$
$$L_2^* = \{[g,g]^2, [f,h]^3, [f,f,h,f,g,h,g,f,h]^8, [g,h,f]^2\}$$

their individual analyses result in the discovery of two different and independent BP models (see Figure 2.18). Both of these models BP_1 and BP_2 are more precise (see precision quality criterion in Section 2.4.1) with regards to their respective source logs L_1^* and L_2^* than the overall BP model (see Figure 2.12). Additionally, instead of only discovering one overall representative process like in the traditional process discovery problem (see Figure 2.12) the process's evolution and thus arguably a more accurate reflection of the reality recorded in L_1 is discovered. Traditional process discovery algorithms as discussed in Section 2.4.2 work on the assumption that the log describes the behaviour of one single, not changing business process and thus treat every trace/instance within a log with equal importance irrespective of their occurrence timing. The result of these algorithms is an "averaged" model that tries to represent all behaviour captured in the event log even if this behaviour is contradictory. Note, that this is not the case in the concept drift example from Figure 2.18 - here the behaviour is exclusive and can easily be merged.

Because only few real-life processes are in a steady state (as assumed by the traditional process discovery algorithms), detecting, understanding, and dealing with concept drift in the domain of process discovery is of "prime importance for the management of processes" [251], i.e. the problem of concept drift is an important challenge to process discovery algorithms [233, 251]. Since concept drift describes the change of an underlying concept over time it can be translated into a one-dimensional clustering problem based

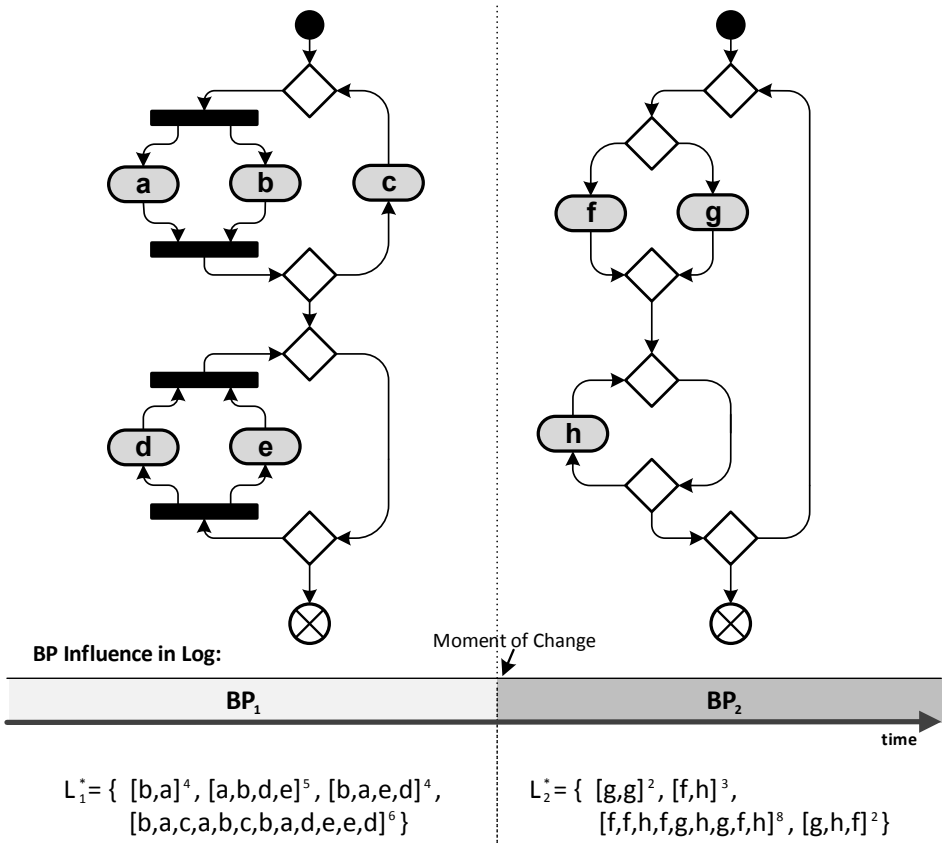


Fig. 2.18 Example Concept Drift in the Original Log L_1

on time, i.e. grouping of instances based on their timestamps into a specified number of clusters which can be interpreted into well-fitting and well-structured BP model versions. The difficulty is to find the right balance between fine-grained and coarse-grained clustering. An extreme example for a too fine-grained clustering is when each trace is considered its own cluster, i.e. represents a BP model (similar to the paths in the flower model in Figure 2.13). Examples of too coarse-grained clustering are the traditional process discovery algorithms for which all instances in the log belong to one single cluster. The challenge of concept drift has gained more attention recently driven by specifically dynamic real-life scenarios, e.g. in the domain of health care, where processes are weakly structured (many exceptions) and inductive methods such as the abstraction-based algorithms have difficulties to describe the behaviour with one single standardised BP model [248]. Here, the discovery of "temporal patterns" produces simpler models and helps to understand time-separated parts of a log [248].

One important and comprehensive step towards a definition of the problem of concept drift in process mining and its accompanying challenges is provided by Bose (et al.) in [20, 21, 23]. In the work three main challenges of concept drift have been identified [20, 21, 23]:

- *Change Point Detection:*
  The challenge of detecting when, i.e. where in the log, a process change has taken place.

- *Change Localisation and Characterisation*:
  If a change has been identified it should be further specified, i.e. what exactly has changed (localisation) and how has it changed (characterisation). With regards to the localisation Bose et al. differentiate between three different perspectives: Control-flow, data, and resource (similar to what was defined in Section 2.1, page 16). Furthermore, the exact location or region within these perspectives should be identified. Another characterisation of a BP concept drift is its "nature" [23]: (1) *sudden drift*, i.e. an immediate substitution of a complete BP by another at one single point in time, (2) *gradual drift*, i.e. for a limited time both BPs coexist during the substitution of one BP by another, (3) *recurring drift*, i.e. a set of different BPs are substituted back and forth (alternating BPs, e.g. to adapt to seasonally recurring circumstances), and (4) *incremental drift*, i.e. a drift from one BP to another by smaller incremental intermediate BPs. Note, that this classification is not mutually exclusive, e.g. recurring and incremental drifts can be either sudden or gradual. While in the BP flexibility Section 2.3 change from an enactment point of view was discussed, the concept drift characterisation of change is from an diagnosis point of view. For instance, modification policies like "Migrate" or "Flush" respectively represent sudden or gradual drifts. The characterisation from Bose et al. is an essentially a different view derived from the BP flexibility domain. It is, in comparison, a simplified view because certain aspects of an enacted change can not be extracted from the log, e.g. swiftness or anticipation, or are of no relevance to the concept drift problem, e.g. momentary change does not constitute a concept drift but should be considered noise.

- *Change Process Discovery*:
  Is the challenge to describe a discovered change process in the second-order dynamics [23], i.e. to discover possibly existing higher-level patterns that specify the order and other specifics of recurring changes. For instance, due to seasonal circumstances concept drift patterns could be modelled as a usually non-concurrent process.

Furthermore Bose et al. see two different classes of concept drift analyses on an event log: *Offline analysis* represents a scenario where the concept drift discovery on an event log can be performed without any real-time constraints and *online analysis* which describes the scenario where a concept drift has to be discovered in (near) real-time. Some solutions for the latter type are also discussed in the following Online Process Discovery Section 2.4.4.

With regards to the first type of offline concept drift discovery Bose et al. propose a solution to address the first and partly the second of the identified challenges in [21]. Though restricted to the control-flow perspective, the proposed approach is able to detect points of change in time and localise these changes in the control-flow. In the work the log is split up into a number of different parts. For each of them four measures are

calculated upon which a change discovery can be achieved [21]: For each of the activities the *relation type count* triple is calculated by determining how many times activities always, sometimes, or never eventually (not directly) follow the specified activity. Considering L_1^* the triple is $(0,2,3)$ for activity d because d is never followed by any activity in *each* of the instances ($\rightarrow 0$), d followed by activities d and e in *some* of the instances ($\rightarrow 2$), and d is *never* followed by three of the involved activities a, b, and c ($\rightarrow 3$). This measure can also be calculated for the "precedes" relation, i.e. the number of times an activity is always, sometimes, or never preceded (not directly) by the other activities. A second measure calculated is the *relation type entropy*, i.e. entropy over the relation type count vector. In contrast to relation type count and entropy which are global measures over each instance in the entire (sub-)log, Bose at al. also suggest two local features [23]: *Window count*, which is the number of times a specified activity is followed by another one within a certain window, and *J-measure*, the probability of a particular activity being followed by another within a certain window, calculated based on the window count values.

These four "feature" values can then be used to identify a concept drift. This is achieved by splitting up the (time-ordered) log into sub-logs of the same length, i.e. contain same number of traces, and compute feature sets for these sub-logs. If significant changes in the feature values from neighbouring sub-logs are identified concept drift can be assumed. In order to find the most probable position for such a drift a *hypothesis test* is executed [23], which can generally be described as a sliding of the sub-log ranges over the log and finding the point for which the feature values differ the most. This method has been evaluated using the methodology of rediscovering, i.e. the position/time of artificially imposed changes recorded in a log were successfully detected [23], but also in a real-life setting by detecting drifts in three different process event logs of a Dutch municipality [22]. However, a few shortcomings are: This method is only effective if a suitable sub-log length is chosen; Changes in more complex constructs like loops can not be sufficiently detected since it is not captured "how many times" an activity eventually follows another activity in one trace; Gradual or incremental drifts are hard to detect with a fixed sub-log length, e.g. the specified length can at the same time both be too small and too large to detect some of the incremental intermediate BP changes.

The problem of concept drift does not only present a challenge for process discovery but can also help to solve it. One such approach is explained in the work by Weber et al. [275, 276]: Here the α-algorithm [249] as introduced previously in Section 2.4.2 is used to create Petri Nets and transform them into probabilistic deterministic finite automata (PDFA) [260, 261]. Similar to the approach from Bose a hypothesis test is carried out which involves the usage of a sliding window to extract the individual sub-logs which are examined for concept drifts after transformation to PDFAs, i.e. compare the distribution generated by the discovered PDFA with the "ground truth". This is essentially carried out for every new trace and it is argued that the approach complies with real-time constraints because it guarantees a result within a pre-specified amount of time (regardless

of how long that time is). To achieve this upper bound run-time for each iteration the approach uses an algorithm presented in [274] to determine the required trace length necessary to, with a certain probability, correctly (re-)discover the structure of a Petri Net model. The detection of concept drift in this approach is mainly focussed on probabilities and not on the structure of the Petri Net [275]. Also, due to the possibility of infinite states it is restricted to acyclic models.

Another approach by Carmona et al. in [34] which extends the initial process discovery approach introduced in [33] to detect concept drift by utilising the theory of *abstract interpretation* [41]. First an abstract representation in the form of a polyhedra is built for some initial traces in a log. Subsequent traces are examined whether or not they lie within the initial polyhedra. If so, it is considered to be a trace from the same process. If a trace lies outside of it a changed process is indicated and the polyhedra is updated. However, this technique is again a factual analysis that is not able to handle noisy behaviour, i.e. to "ignore" infrequent behaviour.

Generally, all of these approaches can be seen as a type of pre-processing with which the log is split up into different parts that are then individually analysed to discover the different versions of a BP as models. The change point information identified may also help to address the third challenge of concept drift: To discover the change process. In [88] it is examined how better decision support can be achieved for flexible processes by using information about the processes' changes. For this two different process mining algorithms, multiphase miner [255] and an adapted method by Cortadella et al. based on the theory of regions [40], has been used in order to identify a change process. The process is discovered from a so called *change event log* that is provided by the adaptive workflow management system ADEPT$_{flex}$ [191] (see Section 2.3.2) and stores detailed context information about each change that was applied to the workflow. These change logs, however, are in the most cases not available and thus detailed information about the changes/concept drifts are not known at the time of analysis. The technique of Bose et al. can be a substitute for analysing these changes but being an a-posteriori analysis the information about the change characteristics lacks in detail as opposed to logged information at the time of the enactment of the change.

2.4.4 Online Process Discovery

Traditional process discovery algorithms, as introduced in Section 2.4.2, are static, offline approaches that analyses an historical event log. Their main goal is to improve the quality of the discovery results, i.e. optimising simplicity of the model and conformance with the input log (see Section 2.4.1). An alternative to this approach is the online approach based on Complex Event Processing techniques (see Section 2.2.3): Immediate processing of events when they occur to information of an higher abstraction level (in this case BP models). The motivation is to have a run-time reflection of the employed processes based on up-to-date rather than historical information which essentially allows business

analysts to react quicker to changes or occurring bottlenecks etc. in order to optimise the overall performance of the monitored processes. In accordance to this objective online process discovery algorithms have to deal with two additional challenges as opposed to the traditional process discovery algorithms:

1. The application of online process discovery is executed in a real-time setting and thus is required to conform to special memory and execution time constraints. Especially with regards to many modern systems producing "big data", i.e. data that is too large and complex to simply store and process it [138]. This means in particular, that online algorithms should be able to (1) process an infinite number of events without exceeding a certain memory threshold and (2) process each event within a small and near-constant amount of time [27]. For instance, a naïve approach is logging every event and with each new occurrence (or periodically) the entire log is (re-)analysed with a traditional discovery algorithm. This approach does not comply with the run-time constraints for online process discovery since (1) the log grows with each event and thus will exceed the memory threshold after a finite number of events and (2) the run-time of the discovery algorithm will increase for analysing an ever-growing log. Additionally, to satisfy the run-time constraint an online process discovery algorithm needs to run autonomously without human interaction or supervision. This excludes discovery algorithms that require manual effort like the step-wise approach based on theory of region (see Section 2.4.2) [243].

2. Online process discovery algorithms are required to deal with concept drift caused by dynamically changing processes during run-time (see BP Flexibility Section 2.3). As discussed in the previous section real-life BPs are often subject to externally or internally initiated change which has to be reflected in the results of an online process discovery algorithm. Whereas concept drift detection algorithms try to determine the time and location of these changes, online process discovery algorithms are only required to anticipate them in the sense of being reflected in the results. This solves the conceptual difficulty of detecting gradual change where the time cannot be exactly identified (thus making it impossible to split the log) because behaviour of two differing BPs are contained in the current stream of events. On the downside, it makes it more difficult to find a fitting model if the behaviour of the two co-existing processes is conflicting. Generally, online discovery algorithms should be able to (1) reflect newly appearing behaviour as well as (2) forget outdated behaviour.

Although not specifically, *incremental process mining* as introduced in [37] does attempt to anticipate the problem of concept drift to some extent. Here, the assumption is that a log does not yet contain the entire behaviour of the process (i.e. an incomplete log) at the time of the discovery of the initial (declarative) process. Additional behaviour, that occurred after the initial discovery and captured in a second log, is analysed separately and the new information is then added to the existing BP model. This is possible due to

structure of declarative BP specifications. The process of incrementally analysing log segments and then extending the BP model accordingly, i.e. incremental process mining, is motivated by the assumption that the update of an already existing (declarative) BP model is easier than to always analyse the complete log from scratch. One shortcoming of this approach is that only new behaviour is added but outdated behaviour is not removed. Another approach called *incremental worklfow mining* is based on the same principle but does discover and adapt a Petri Net by incrementally processing log segments [119–121]. It is a semi-automatic (and prototypical) approach specifically designed for dealing with process flexibility in Document Management Systems that does not anticipate incomplete or noisy logs. A third incremental approach is presented in [219] which utilises the theory of regions to create transition systems for successive sub-logs and eventually transform them into a Petri Net. Albeit based on a slightly different concept, incremental process mining approaches can be considered for online process discovery: The event processing could be designed to group a number of successive traces into sub-logs which are then individually analysed and incrementally update and extend the BP. However, a conceptual weakness of incremental mining approaches is the lacking ability of forgetting old behaviour and thus not supporting, for instance, *revolutionary* changes as discussed in Section 2.3.

In the context of online process discovery, a synonymous term sometimes used is *Streaming Process Discovery* (SPD). SPD was coined by Burattin et al. in [27]. In their work the HeuristicsMiner [278](see Section 2.4.2) has been modified for this purpose and a comprehensive evaluation of different event stream processing types was carried out. The fundamentals of the HeuristicsMiner remain the same but the footprint in form of the Directly-Follows Relation Count (see Figure 2.17) is dynamically adapted or rebuilt while processing the individual events. From this the Causal Net (or the Petri Net with an additional transformation) is periodically extracted, e.g. for every event or every 1000 events, using the traditional HM methodology. For instance, for the evaluation of the different streaming methods the HM discovery was triggered every 50 events [27]. Three different groups of event streaming methods have been implemented and investigated:

Event Queue: The basic methodology of this approach is to collect events in a queue which is representing a log that can be analysed in the traditional way of process discovery. Figure 2.19 shows three basic types of this methodology: (1) In the *sliding window* approach the queue is a FIFO (First-In-First-Out), i.e. when the maximum queue length (queue memory) is reached for every new event inserted, the oldest event in the queue is removed. The picture shows the development of the log/queue for each triggered discovery analyses. (2) In the *periodic reset* approach the queue is reset whenever the maximum queue length is reached. (3) The *uniform split* represents a special case of the sliding window or periodic reset approach when the queue memory equals the discovery frequency. This approach was not analysed by Burattin et al. but was included into the figure as a naïve baseline scenario. Note, that the approaches using the hypothesis test for the con-

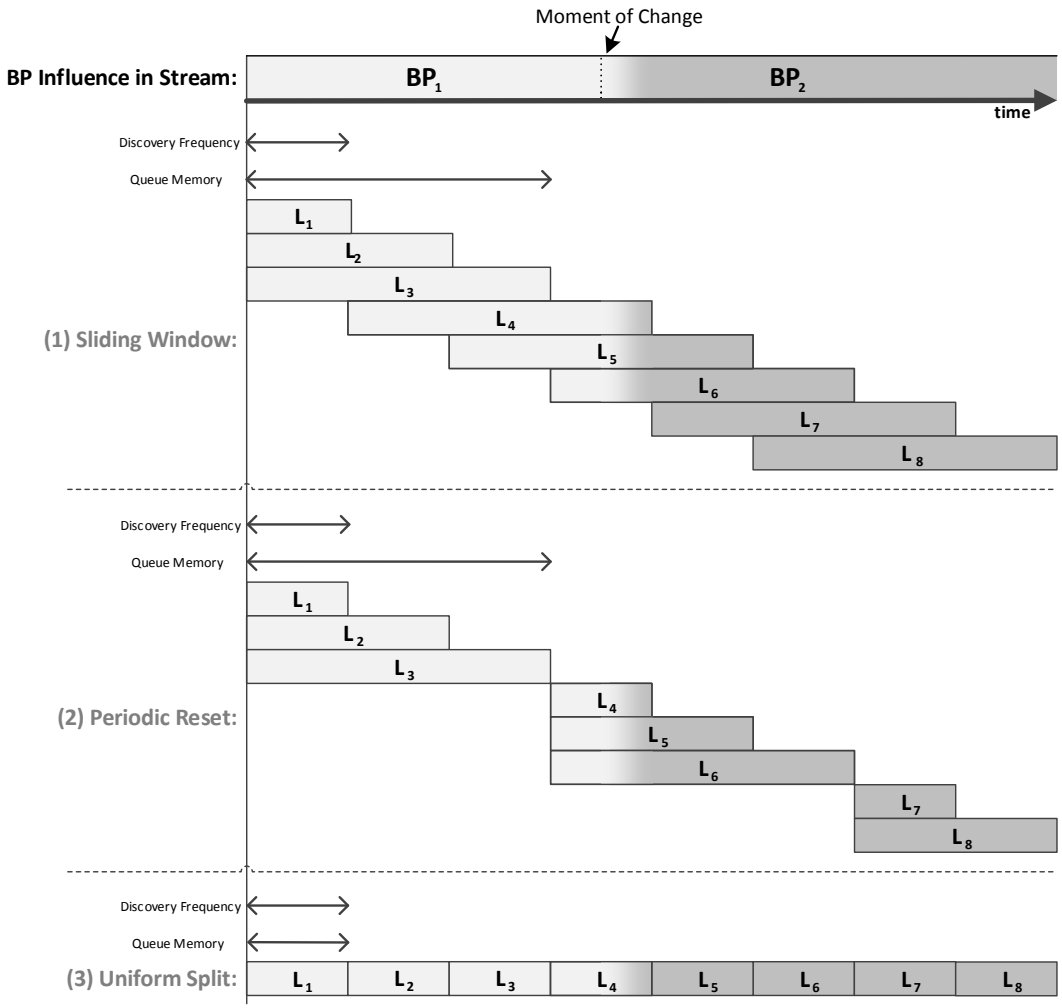


Fig. 2.19 Different Types of Event Queues [27]

cept drift detection algorithms, e.g. [23, 276], are based on a (shifting) uniform log split. The main advantage of these approaches is that the event queue can be regarded as event log and allows for other discovery/mining analyses on event logs such as mining of the performance perspective. Two of the main disadvantages are: Each event is handled at least twice: once to store it in the queue and once or more to discover the model from the queue; Also, it does only allow for a strict interpretation of "history" either the event is still in the queue or not. In the first case an older event has the same influence as a newer event. Essentially, the event queue approach is a simple method to use offline process discovery solutions to solve the online process discovery problem.

Stream-specific Approaches: Stream-specific approaches already process events into footprint information, i.e. queues of a capped size hold information about the latest occurring activities and directly-follows relations. When a new event occurs all values in the queues are updated and/or replaced. Burattin et al. distinguish between the following three update operations: (1) *Stationary*, i.e. the queues function as a "sliding window"

over the event stream and every queue entry has the same weight, (2) *Ageing*, i.e. the weight of the latest entry is increased and the weights of older entries in the queue are decreased, and (3) *Self-Adaptive Ageing*, i.e. the factor with which the influence of older entries decreases is dependent on the fitness of the discovered model in relation to latest events stored in an additional sample queue of a fixed size: quickly decreasing for a low fitness and slowly decreasing for a high fitness. Generally, stream-specific approaches are assumed to be more computation-balanced since events are only handled once and directly processed into footprint information (as opposed to the event queue approaches where the discovery cycle contains all computation-heavy algorithms) [27]. Burattin et al. also argue that ageing-based approaches have a more realistic interpretation of "history" since older events have less influence than newer events. One disadvantage is that the footprint is captured through a set of queues with a fixed size: if this size is set too low behaviour is prematurely forgotten (i.e. removed from the queue); if this size is set too high you might never forget certain information. The effect of a too large queue length is mitigated for the ageing HM approaches since their weight continuously shrinks and is eventually deemed irrelevant noisy behaviour by the core HM algorithm.

Lossy Counting: Lossy Counting is a technique adopted and modified from [139] that uses approximate frequency count and divides the stream into a fixed number of buckets.

The evaluation of these methods was carried out for three example processes of different sizes (8-19 activities) but without looping constructs involved. These were simulated and two of them were modified during execution at specific points in time. The quality of the results was measured using the criteria outlined in Section 2.4.1: Fitness [3], precision [155], generalisation, and simplicity. The conformance measures fitness, precision, and generalisation were computed on the basis of a queue of a fixed size containing the last events representing the sample log (similar to the sliding window approach). The conclusions, however, were mostly drawn with regards to the fitness(recall) and precision criteria. Additionally, computation time and memory usage were investigated. Summarised, the evaluation yielded the following results: With regards to the computation-time and memory consumption the streaming-specific approaches (stationary, ageing, self-adaptive ageing) were outperforming the event queue approaches (sliding window, periodic reset) especially for higher queue lengths and the lossy counting approach. However, only the time for processing an event was investigated, not the time needed for the discovery of the process. The self-adaptive ageing, for instance, would require more time since conformance criteria have to be computed to adapt the ageing factor. Considering the conformance criteria, all approaches performed similarly well if the correct parameters were chosen (e.g. ageing factor, queue lengths, number of buckets) for stationary streams and converged to the best fitting BP after some time. For streams with concept drift the streaming approaches and lossy counting performed better than the event queue approaches. For the larger example the stationary streaming approach (with the same

weights) achieved a significantly higher fitness which can be explained with the experiment setup: The conformance checking treads every event in the sample queue equally independent of their "age".

Another approach for discovering concept drifts on event streams of less relevance to the thesis' topic is presented in [138]: A discovery approach for declarative process models using the sliding window approach and lossy counting to update a set of valid business constraints according to the events occurring in the stream.

2.5 Performance Decision Support for Business Processes

An overview and examples of different types of process analysis and reasoning in the context of process intelligence, decision support, and business activity monitoring are provided in Section 2.2.3, e.g. what-if analysis, performance monitoring/extraction, performance prediction, process discovery, etc. Some are applied online in an automated fashion (e.g. performance monitoring, notifications) and others offline and may require human interaction (e.g. what-if, sensitivity analysis, process discovery). All of which, however, are motivated by potentially improving the decision process for (re-)designing a business process (see Figure 2.7 on page 24) [70]. Generally, the decision making in the BP-domain (and other domains) is based on rules, constraints, and goals. Business analysts strive to optimise the performance of a business process with regards to the specified goals while still conforming to the rules and respecting the constraints. The performance of a process is usually expressed by quantifiable measures against which the goals can be evaluated. Thus, performance-centric decision support for BPs describes techniques that provide additional, computed performance measures to support the business analyst in making decisions about the business process in question. These measures essentially enable the analyst to understand the "health" of a process, obtain detailed insight into the process execution as well as its environment, and react if an adaptation or exceptional interference becomes necessary, e.g. reallocation of resources or prioritising parts of a process.

In this section performance-centric decision support techniques with the focus on performance analysis and prediction are discussed. In the context of the thesis' topic run-time performance prediction is used twofold: As means (1) to conceptually prove the applicability of the devised reasoning approach based on descriptive business process models at run-time (see Section 6.1) and (2) to evaluate the hypothesis that by employing this DBPMRT approach better higher-level reasoning (i.e. prediction) results can be achieved (see Section 6.2).

2.5.1 Process Performance Indicators

Measures reflecting the health of the organisation and its processes are usually called *Key Performance Indicators* (KPIs) and are derived from higher-level business goals [282]. In

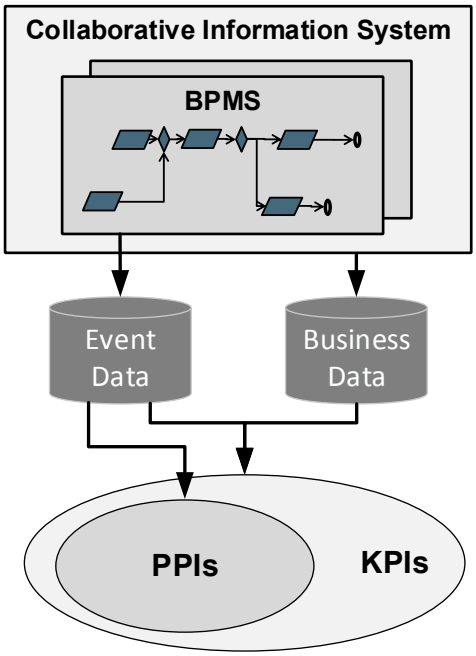


Fig. 2.20 The Relation between PPIs, KPIs and the System

some literature, KPIs that are directly related to the performance of a business process or its elements are called *Process Performance Indicators* (PPIs) [49, 50, 56, 150, 179]. In [50] a PPI is defined as *"...measure that reflects the critical success factors of a business process defined within an organisation..."* which *"...focuses exclusively on the indicators defined on the business process"*. The conceptual relation between PPIs, KPIs, and the system/organisation is shown in Figure 2.20: Whereas PPIs are exclusively calculated on the basis of process-specific data (usually event data), KPIs may make use of other sources, e.g. business or accounting data. That is, the set of PPIs is a specialised subset of the set of KPIs.

PPI measures can be very diverse, for instance, be generic (e.g., *instance occurrence of process*) or process-specific (e.g., *net duration of activity "Credit Check"*), calculated for a single instance (e.g. *current duration of a process instance*) or aggregated over several/all instances (e.g., *average duration of process*), or differ in many other aspects, e.g describe resources or control-flow elements, etc. A comprehensive overview of over 500 performance measures and their dependencies in the domain of software processes is provided by Monteiro et al. in [150]. A more generic view on PPIs is given in [50]: Here, an ontology is proposed that allows for generic modelling of PPIs in order to integrate the managing of PPIs into the BPM lifecycle. Furthermore, three different types of measures are distinguished in [50]: (1) *base measures*, i.e. extracted from a single process instance using a specified method (count, time, condition, data), (2) *aggregated measures*, i.e. calculated by applying a certain aggregation function (min, max, sum, average, count) on a set of base measures belonging to different instances, and (3) *derived measures*, i.e. applying a customised mathematical function on one or more measures (base or aggregated). One shortcoming is that frequency (e.g. process instance frequency), probability (e.g. path

probabilities for decision), and resource-related (e.g. resource utilisation) performance measures are impossible or difficult to model, i.e. only via derived measures. Similar attempts following the approach of PPI modelling for automated process monitoring exist, e.g. [68, 130, 149, 282].

These BAM solutions are in the context of process mining usually classified as *model enhancement* (see Figure 2.9 on page 26) because basic information about the process is already provided, i.e. the process definition/model is a required input to link the modelled PPIs to the respective BP elements. These approaches are designed for stationary BPs. For dynamically changing processes, PPIs cannot be modelled beforehand but have to be of a rather generic type, automatically adapting to the underlying BP which may change over time. In the following a selection of these generic PPIs are listed [69, 70, 189]:

- *Process Instance Occurrence* describes how often a start event of the process is triggered within a certain time period (as frequency) or how much time passes between the start of two instances (as duration).
- *Activity Net Working Time* specifies the time needed (for a resource) to process the specified activity. This is the effective time the resource takes to fulfil the task and does not include delays, e.g. unavailability of a suitable resource.
- *Activity Processing Time* specifies the overall time needed to process the specified activity including all delays, e.g. waiting for suitable resource assignment.
- *Throughput* is the count of how instances have "passed through" this BP element during a specified time period.
- *End-to-End Processing Time* determines how much time it takes to carry out the whole process from start to end.
- *Path probabilities* describes the likelihood of the exclusive paths following a decision element.
- *Utilisation* specifies the percentage of time a resource or set of resources (i.e. grouped as role) was busy, i.e. effectively processing tasks.
- *Queue Length* is a count describing how many tasks are currently in the queue and waiting to be processed by a resource with the appropriate role.

If a performance measure is an average aggregate over multiple instances and the actual values of the instances deviate from the average it is common to capture not just the average but also the *standard deviation* (to form a *Normal Distribution* [104]) or the *z-value* (to form a *Confidence Interval* [24]) in order to reflect the probabilistic nature of the PPI. For example, process instance occurrence is on average 10 per minute but does actually range from 5 to 16 per minute. Duration or frequency-based PPIs such as process instance occurrence, activity net working time, activity procesing time, and end-to-end processing time are usually recorded as normal distributions rather than simple average values.

2.5.2 Process Performance Prediction

The PPI values and their evaluation against their target values support the process of decision making in the BP lifecycle. Furthermore, they can be the basis for automatic higher-level decision support techniques such as bottleneck detection (e.g. by examining the average duration of a resource being active in comparison to all other resources [195]), notifications for SLA violations (e.g. end-to-end processing time not within the agreed margins of the service agreement [285]), or predictions of PPIs. The focus of this thesis is on performance prediction which is the basis of pro-active decision support and many higher-level analyses, e.g. what-if, sensitivity analysis, and optimisation.

A first type of performance prediction techniques is represented by analytical or statistical approaches. Business applications for PPI monitoring (e.g. *SAP Operational Process Intelligence* [206]) or data-centric Business Intelligence (BI) solutions (e.g. *SAP BusinessObjects BI Platform* [205]) often use trend analysis techniques which are based on extrapolations from the historical data of the performance measure [24]. For this regression algorithms can be employed to fit a type of function (e.g. linear, polynomial) into a set of data points in a way that the distances between the data points and the function are minimal. The retrieved function can then be used to calculate future values, i.e. extrapolation. This approach, however, is solely based on a time series of values and does not take information about the business process into account. Other analytical techniques exist where the BP is represented by a mathematical model, e.g. FMC-QE [178]. The biggest advantage of this type is that instant results can be computed, which is why analytical techniques are preferably used in high-level analyses like optimisation where thousands of different cases have to be analysed as fast as possible. Disadvantages are (1) that they typically are only simplified approximations (e.g., conditional loop behaviour hard to be represented by a formula [178]), (2) impose additional constraints and are difficult to use [29], and (3) do not take information of the current state of the BP into account, i.e. they can essentially only be used for steady-state predictions (such as what-if predictions).

Another group of techniques for process performance prediction is based on the simulation of business processes. *Simulation* "... attempts to mimic real-life or hypothetical behaviour" [232]. Simulations can be classified in several ways [193]: They can either be (1) *continuous* or *discrete*, (2) *deterministic*, or *non-deterministic*, (3) analyse *dynamic* or *static* behaviour (the latter is also referred to as *steady state analysis* in many publications, e.g. [201, 210, 232, 235, 244, 252]), and (4) in the case of a discrete simulation implemented as *time-slicing* or *discrete-event* approach. According to [193], the discrete-event approach is very common in event-based simulations and in commercial tools like *AnyLogic* [6].

In the BP domain simulations are usually discrete and non-deterministic utilising the discrete-event simulation approach. For this *"...the system is modeled as a series of events, that is, instants in time when a state-change occurs."* [193]. One implementation of the discrete-event simulation is the *three-phase approach* [173] shown in Figure 2.21. It is

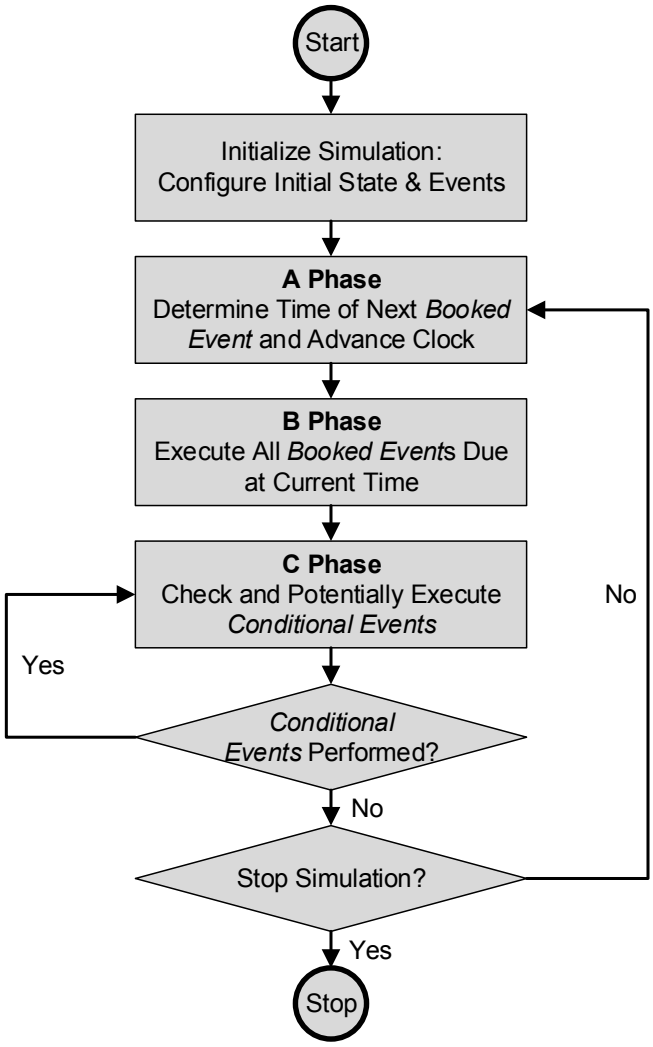


Fig. 2.21 Algorithm of the three-phase discrete-event simulation approach [193]

based on two kinds of simulation events[9]: *booked events*, i.e. state changes scheduled to occur at a certain time, and *conditional events*, i.e. state changes that are executed if a condition in the simulated system becomes true. With regards to a simulation of business processes, booked events are commonly used to register activity completions or instance occurrences and conditional events are used to express the semantic of the process. The execution of any of these simulation events change the state of the system simulated and/or schedule new booked events. In the *A phase* of the three-phase algorithm (see Figure 2.21) the time of the next event is determined by examining the event list and the simulation clock is advanced to that point in time. In the *B phase* all booked events are executed which are due (for the current time). In the third step, the *C phase*, all conditional events are checked and, if true, executed. Since the successful performance of a conditional event alters the state of the system, thus possibly enabling another conditional event, the C phase is repeated until no further conditions are true. In order to not unnecessarily slow down the simulation the number of conditional events should be kept small or completely avoided since all of them have to be checked repeatedly at every point

[9] Simulation events are not the same as BP events discussed in previous sections

in time. The three steps are continuously repeated until the simulation's stop condition is true, e.g. when the predetermined time is reached.

In the domain of business process simulation two different specialisations of BP simulations can be distinguished and will be discussed in the remainder of this section: *steady-state simulations* and *short-term simulations*.

Steady-state BP Simulations

The method of simulation is considered to be versatile, impose only a few constraints, and produce results that can be similarly interpreted as the ones of the simulated system [232]. This is why BP simulation is one of the most established techniques in the domain of BPM supported by many tools, e.g. [72, 131, 180, 189, 200, 290]. Most of these tools focus on analysing *steady-state* situations which are simplified models that help investigating the future performance of a BP variant on a strategic or tactical level [232]. Steady-state simulations are usually associated with the (re-)design phase of the BPM lifecycle (see Figure 2.7 on page 24) and can be used to investigate the performance of a BP variant before enactment, i.e. for what-if analysis. As an input steady-state simulations require the following information [232, 292]: (1) the business process structure, i.e. control-flow, resource and role information, and (2) annotated performance and routing information, i.e. the incoming work/arrivals, activity details (e.g. type, time to process), and routing details for the decisions in the control-flow. In relation to the PPI examples listed in the previous Section 2.5.1, the performance and routing information corresponds to the three PPIs *process instance occurrence*, *activity net working time*, and *path probabilities*. In [72, 73, 131, 180, 189] solutions following the approach of a steady-state BP simulations are proposed.

An approach that supports a more dynamic interpretation of BP simulation is introduced in [210]: Here an adaptive simulation model with a history-dependent mechanism that propagates changes through the model is proposed, i.e. it assumes that the performance parameters driving the simulation change over time are dependent on the development of other parameters. These performance interdependencies are learnt from the historical data stored in the event log. Note, that other approaches also allow for simulation parameters changing over time, e.g. [72, 189], however, they have to be modelled and are not extracted from historical data. Furthermore, all of the above simulation solutions do typically start "empty", i.e. without any active traces, built-up queues, etc., and need some time to "warm-up" until realistic measurements can be achieved, i.e. the steady-state is reached.

Short-term BP Simulations

Since the current state of the system is irrelevant for steady-state simulations, they are not deemed suitable for operational decision support at run-time [200]. In contrast, short-term simulations take the current state of the system into account and are designed to al-

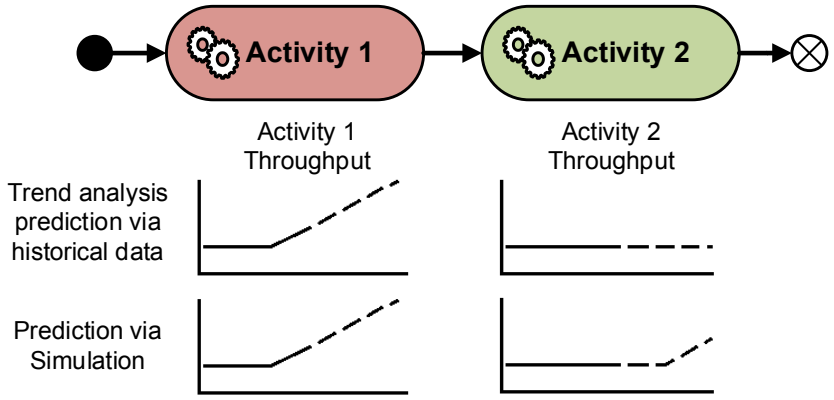


Fig. 2.22 Short-term Prediction via BP Simulation vs. Trend Analysis

low a look into the system's near future, similar to pressing the *"fast forward button"* [232]. In essence, the objectives of both types of simulations are different: Whereas steady-state simulations are to investigate what-if scenarios, i.e. provide answers on a *strategical* and *tactical* level, short-term simulations help to predict problems and bottlenecks that will occur in the near future in order to allow for pro-active mitigation before they take effect, i.e. provide answer on an *operational* level [232]. Since short-term simulations are built on the current state of the system they are also called *transient simulations*. When comparing short-term prediction approaches, a transient simulation is expected to provide better results than analytical analyses based on the historic data of the measure (e.g. trend analysis) because BP simulations take additional structural information into account (control-flow, resources). This is conceptually highlighted in Figure 2.22 for a simple sequential BP example: While the trend of the throughput of "Activity 2" is flat (upper right graph), the simulation based on structural information ("Activity 1" is followed by "Activity 2") and the current state (there are currently an increasing number of BP instances processed at "Activity 1", i.e. throughput for "Actitivy 1" is rising) will correctly predict a later rise of the throughput of "Activity 2" (lower right graph). Other such dependencies between PPIs can be hidden in a BP model and thus can be predicted more accurately with a short-term simulation, e.g. the sharing of resources by two or more activities.

To the best of the author's knowledge only a small number of publications exist that address the challenge of short-term BP simulations for operational decision support [200, 201, 220, 290]. Solomon et al. show conceptually in [220] that simulations can predict KPIs and essentially help adapting the running process. In their work combinations of one-step predictions (always just the next KPI value is predicted) are compared for their performance on a simple model of three activities. It is, however, neither stated how all the information for the simulation model is extracted from the monitored system, nor how it is integrated or modelled. In [290] it is conceptually shown that simulation models containing design, performance, and state information can be used to create a simulatable YAWL [245] model (see Section 2.2.1). Rozinat et al. extend this work in [200] and [201] arguing that some of the information of the YAWL simulation model can be

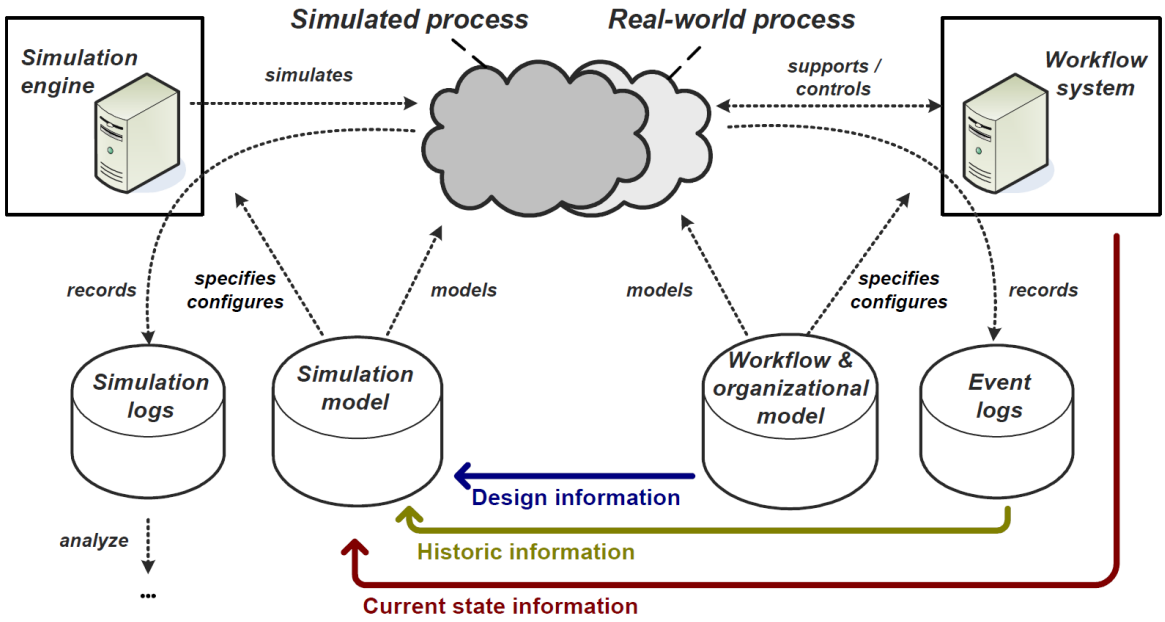


Fig. 2.23 Short-term Simulation System Integrated with the Workflow System [200]

extracted from the event log and the workflow system. Figure 2.23 shows the integration of the short-term simulation system and the worklfow system as well as the involved sources and types of the information of a simulation model. It is distinguished between three types of information [200]: *design information*, i.e. modelled control-flow and resource perspective, *historic information*, performance parameters availability patterns of resources extracted from the log, and *state information*, i.e. obtained from the worklfow system are the progress and associated data of currently active instances as well as current resource availability (busy or not). When this model is simulated a log is created containing the simulated events of the near future. Similar to the original event log, this simulation log can then be analysed for its performance, i.e. predicted PPIs. The assumption for this approach is that both, workflow and simulation system, use the same static workflow model and states can be easily extracted from the system.

The biggest disadvantage of simulations is that they are time consuming and not very scalable: size of the business process, time to simulate, and average instance occurrence similarly have a linear influence on the execution time of the simulation. Furthermore, BP simulations (short-term and steady-state) are non-deterministic: Decisions in simulations are often not based on data but on probability distributions; Activity net working times are based on distribution values, e.g. normal distribution; Resource appointments are random to some extent. For this reason BP simulations should be executed several times to gain a certain confidence about the results (either recorded as average value or as confidence interval) [104, 193]. This is true for all steady-state analyses but some of the presented short-term-predictions base their prediction results on one single simulation run, e.g. [200, 220].

2.5.3 Performance Prediction from Event Logs

In the previous section it was discussed what different prediction techniques exist (analytical vs. steady-state simulation vs. short-term simulation) as well as their goals, advantages/disadvantages, and what input information these techniques require. In contrast, this section is concerned with state of the art performance prediction techniques that do not use any modelled information but are solely based on data contained in the event log.

A first type are approaches using analytical techniques to predict the future behaviour on the instance level. One such approach is presented in [252]: Here a non-parametric regression model is used to predict the time currently active BP instances still need to finalise. Another approach that focuses on predicting the completion of BP instances is presented in [241] and extended in [244]. The approach is based on the extraction of a transitions system from an event log utilising the state-based process discovering approach presented in [243] (see end of Section 2.4.2), annotating the resulting transition model with time information computed from earlier finished instances, and use this to only predict the completion time of the not finished instances. Neither of these approaches take the resource perspective into account. Furthermore, both are limited to predict completion times for open traces but do not provide the possibility to predict aggregated PPIs as introduced in Section 2.5.1, e.g. end-to-end processing time, utilisation.

As established in the previous section, simulation is a more versatile approach to predict the development of a BP's future performance. In [197] it is shown that BP simulation models (in the form of Coloured Petri Nets (CPNs) [109]) can be extracted from the event log (using the α-algorithm [249] as discussed in Section 2.4.2) and annotated with decision, time, and probability data that is also extracted from the event log. This is the minimum of information needed to carry out a BP simulation on a rather simplified BP interpretation. This work was extended in [198] to include information about the resources and thus allow for more accurate prediction results. Figure 2.24 shows the involved agents and models of this approach [198]:

1. First, the control-flow is extracted (again with the α-algorithm) to build a Petri Net.

2. Based on this and the event log the Petri Net is enhanced with performance values and data dependencies respectively to build two annotated CPNs. The decision point mining is based on the classification problem and discussed in more detail in [199]. Note that for decisions both, probabilities and data dependencies, are computed which is redundant but allows a choice for the user to simulate on the basis of either of them.

3. Furthermore, the resources and their respective roles are discovered from the event log. For this the "metrics based on joint activities" proposed in [242] is applied. The approach works on the assumption that people/resources that are *"...doing similar things are more closely linked than people doing completely different activities"* [198].

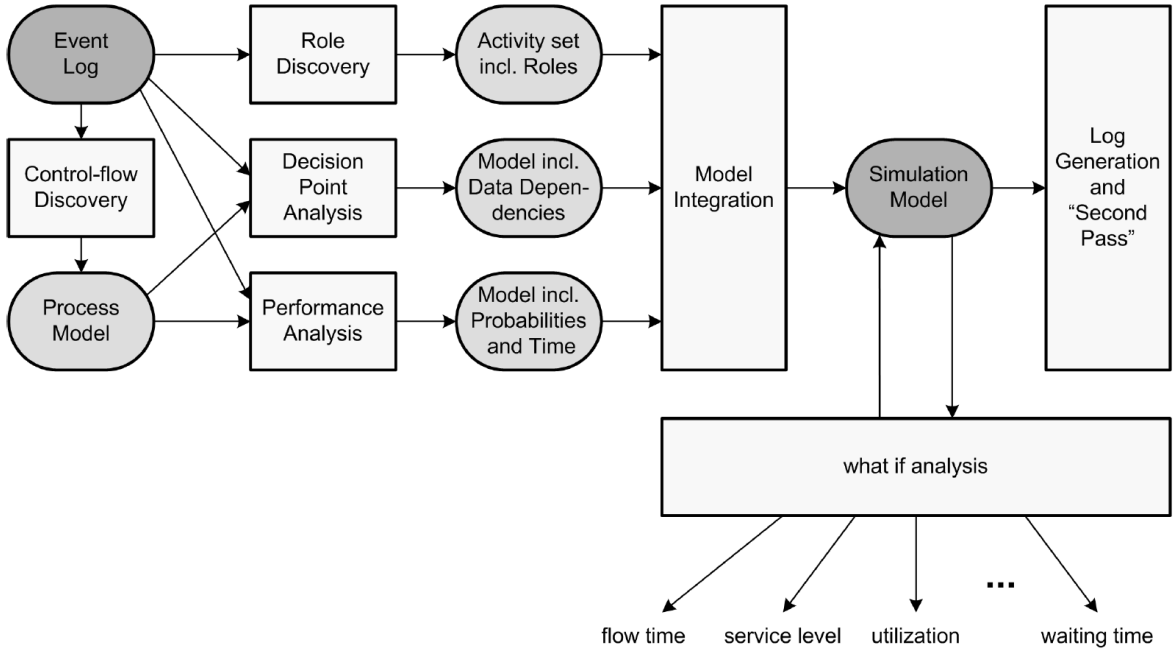


Fig. 2.24 Mining Simulation Model for Steady-State Prediction [198]

The "distance" between two different resources is measured using *Pearson's correlation coefficient* on the two respective profile vectors that contain their association frequencies with all involved activities. The computed distances are the basis for the subsequent clustering algorithm that determines the role(s) of each resource. Other approaches that (partly) address the discovery of the organisational perspective are based on (1) decision trees for staff assignment [136], (2) the handover of roles, an approach based on partitioning and merging of sets of activities based on the assigned resources [28], (3) characterisation of resource availability, an attempt to model human resources more accurately, e.g. taking into account a resource is usually not fully assigned to only the observed process or that activities are processed in "chunks" [239].

4. In a final step the two annotated CPNs as well as the resource model are merged to a final simulation model that can be interpreted and used by a CPN simulation engine.

In none of the discovery approaches for simulation models discussed in the above paragraph the information of the current state is taken into account, i.e. they can only be used for steady-state simulations. The only state-based approach for short-term simulation that discusses the current state as an input is [200] (see Section 2.5.2; Figure 2.23): Here a plugin is proposed that exports the current state from the BPMS (i.e. a YAWL system) into a file of a specialised format that is interpretable by the simulation engine. However, to the best of the author's knowledge no publication discusses how to effectively extract the instance state of the process directly from an event log or monitor an event stream.

2.6 Model-Driven Engineering

Model-Driven Engineering (MDE) is a technique to manage the increasing complexity of modern software platforms and is based on lessons learnt in the past 30-40 years of Computer-Aided Software Engineering (CASE) and object-oriented technologies [65]. It describes the systematic use of models as development artefacts throughout the software lifecycle [115, 209]. In contrast to object-oriented development which follows the principle of "Everything is an object", MDE is the application of the principle "Everything is a model" [14]. This shift of principles promises an effort reduction for the development and maintenance by working at the model instead of the code level [52]. Bézivin defines MDE in [14] as *"the unification of initiatives that aim to improve software development by employing high-level, domain-specific, models in the implementation, integration, maintenance, and testing of software systems"*. At the beginning of this millennium prominent initiatives, i.e. *Eclipse Foundation*[10] and the *Object Management Group (OMG)*[11], have published a set of standards that apply and support the principles of MDE:

- The eclipse project is based on MDE concepts (among many others) and provides a unified set of modelling frameworks, tooling, and standards implementations, such as *Eclipse Modeling Framework* (EMF) [225] and *Graphical Editing Framework* (GEF) [63].
- *Model-Driven Architecture* (MDA) defined by the OMG is a realisation of MDE principles around a set of standards like MOF, UML, XMI, QVT, etc. [169]. One of MDA's main objectives is the separation of modelled system functionality (i.e. PIM) from its specification for a specified platform (i.e. PSM) in order to control their evolution more independently [176].

2.6.1 Models

Depending on the context the term model can have many different general meanings if looked up in an encyclopaedia. The two following are appropriate in the context of Model-Driven Engineering [14][12]: (1) copy of an object, especially one made on a smaller scale than the original, or (2) a simplified version of something complex used in analysing and solving problems or making predictions. A more fitting and apt definition for models as they are understood in MDE is provided in [64, 224]:

- *"A model has a purpose.*
- *A model describes some entity that exists or is intended to exist in the future.*
- *A model is an abstraction, that is, it does not describe details of the entity that are not of interest to the audience of the model."*

[10] http://www.eclipse.org/org/
[11] http://www.omg.org/
[12] Bézivin adopted these definitions from Encarta Encyclopedia which is not publicly available any more.

In summary, models in the context of MDE can be described as purposeful abstractions or reduced representations of a system. Furthermore, Stachowiak differentiates between three types of information stored or not stored in a model [224]: (1) *abundant attributes*, i.e. additional information in the model that is not part of the original object. An example would be the BP model in Figure 2.2 on page 14: Not only the behavioural and structural information of a process is shown but also the model is enhanced with additional graphical information of where exactly elements are positioned in the picture; (2) *selected attributes*, i.e. information shared by both, the model and the original object. This is the data of importance for the specific purpose of the model; And finally (3) *preteritive attributes*, i.e. ignored attributes of the original that is not required in the model, reflecting the fact that models are abstractions of original objects.

Many different types of models find practical usages in science and economy and help abstracting from a (too) complex reality: statistical models, meteorological models, biological models, ecological models, economical models, etc. Furthermore, computer science may be mainly described as the science of building software models [14]. The usage of models in the domains of computer science and software engineering is beneficial for a multitude of reasons, of which the following two are considered to be most important:

Higher Level of Abstraction:
The most important characteristic of an engineering model is abstraction [213]. Model-Driven Engineering allows to raise the abstraction level for developers in order simplify and formalise the tasks involved in the software lifecycle [92]. When developing software a higher level of abstraction helps to reduce the complexity as well as required effort [8]. Furthermore, a higher abstraction level improves communication and analysis capabilities. That is, models are better suited than (software) code to answer questions asked by different stakeholders which generally are of a higher abstraction level [43]. Thus, they can be used to reflect the intentions of stakeholders more precisely while at the same time ignoring implementation details [8]. Thus, models improve the development of software artefacts by providing information about the consequences before they are actually implemented [135]. A general challenge is, however, to find the correct balance between simplification, i.e. raising the abstraction level, and oversimplification, i.e. neglecting details necessary for a specified purpose [8].

Systematic Reuse of Development Knowledge & Platform-Independence:
Independent from the target platform, models can be used in the software development process to describe principles, processes, interactions, and other domain-specific knowledge. This knowledge can be reused in an explicit and systematic way for other platforms and software of the same domain and with the same purpose [8, 14, 124].

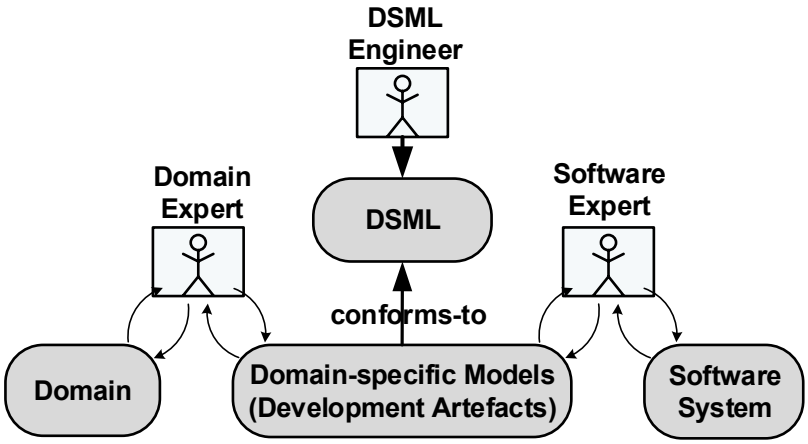


Fig. 2.25 Experts involved in Model-Driven Engineering [69]

2.6.2 Domain-specific Modelling

Domain-specific Languages (DSLs) have to deal with a reduced number of concerns when compared to *General Purpose Languages* (GPLs) [69]. Examples for GPLs are *General Purpose Programming Languages* like *Java* and *C* and *General Purpose Modelling Languages* like UML [114, 165]. DSLs are applied in many different domains, such as grid computing [217], finance [91], or business processes [161, 168, 207]. More extensive DSLs like *Modelica* [170] or *MEMO* [67] offer development support for a multiple different domains.

In [117] the benefits of using DSLs have been quantitatively evaluated with the result that a higher degree of efficiency, e.g. in terms of productivity, can be achieved. In particular, DSLs are beneficial if a family of programs, e.g. the family of business processes, is addressed due to a possible reuse of artefacts. Additionally, languages for a specific domain are typically of a higher abstraction level and thus shorter than their pendant construct in GPLs [114]. As a result, developers using a DSL can concentrate on creative tasks rather than repetitive tasks. The usage of models and DSLs have notably the same two advantages, i.e. reusability and higher abstraction level, as they are essentially two sides of the same coin. In the context of MDE DSLs are also called *Domain-specific Modelling Languages* (DMSLs) and define a consistent set of rules which a domain-specific model has to conform to. With the adoption of DMSLs in the MDE concept the responsibilities can be split up into three different roles (see Figure 2.25) [69]:

- The *DSML Engineer* defines the DSML, e.g. BPMN, and provides tools to create instances of this DSML, e.g. BPMN Modelling Tools.
- The *Domain Expert* uses the provided tools to define and manage her instances of the DSML, i.e. the domain-specific models. Examples of domain experts are *Business Process Analysts* or *Business Impact Analysts*.
- The *Software Expert* is the one who actually implements and configures the development artefacts for a specific system. In some cases this step can be automated.

In this setup the domain-specific models are essentially used as means of communication between domain expert and software expert. With the proposed sharing of responsibilities, domain experts can now be more closely involved in the development process without having to be skilled programmer themselves. The identification and attribute mapping process from the original object to the model is in general called *modelling* or *Domain-specific Modelling* (DMS) in the context of MDE [159]. This is carried out by the domain expert.

The traditional understanding of modelling and models is based on *"...an abstraction of reality in the sense that it cannot represent all aspects of reality"* [196], i.e. modelling is a simplification process. Different to the traditional understanding of modelling, in Model-Driven Engineering models are not only simplifications of reality but formal input and output for computer-based tools and represent implementations of precise operations [14]. To enable a consistent interpretation of models of the same domain they have to conform to the same DSML. Following the "everything is a model" paradigm of MDE, DSMLs are in fact models themselves describing the abstract syntax of domain-specific models and are in MDE literature usually called *meta-models*. To distinguish between relations in MDE, two basic types of abstractions exist [14]:

1. the *represented-by* abstraction is used to define the relation between the original information (or system) and the model, and

2. the *conforms-to* abstraction is to define the relation between a model and its meta-model. Although, in a general sense a model is an instance of a meta-model, the instance-of term is not widely supported in MDE in order to avoid confusion with object-oriented principles.

Since a meta-model is also a model it needs to conform to a *meta-meta-model* which defines the abstract syntax of the meta-model. For this purpose, the Object Management Group (OMG) has defined a standard called *Meta Object Facility (MOF)* [162]. It proposes a four-level architecture, shown in Figure 2.26:

At the bottom level, the layer M0 is the original information, i.e. the real system. An example of such a system in terms of BPM is a enacted implementation of the business process in Figure 2.2 on page 14. Information on level M0 is *represented by* a model at the M1 level. With regards to the former M0 example, the actually executed online order process is represented by the online order BP model shown in Figure 2.2. A model from the level M1 *conforms to* a model in the meta-model layer at level M2. This layer permits modelling tools to operate with M1 models, e.g. editors for DSLs, such as the *Process Composer* of the *SAP Netweaver BPM* [287]. These tools require a formal specification of the abstract syntax of the M1 models they are handling. This specification is defined by the meta-model. Since the Online Order BP conforms to the BPMN language, BPMN is the BP's meta-model and an example for a model in the meta-model layer M2. The abstract syntax of meta-models is defined by a meta-meta-model at the M3 level. Additionally, a

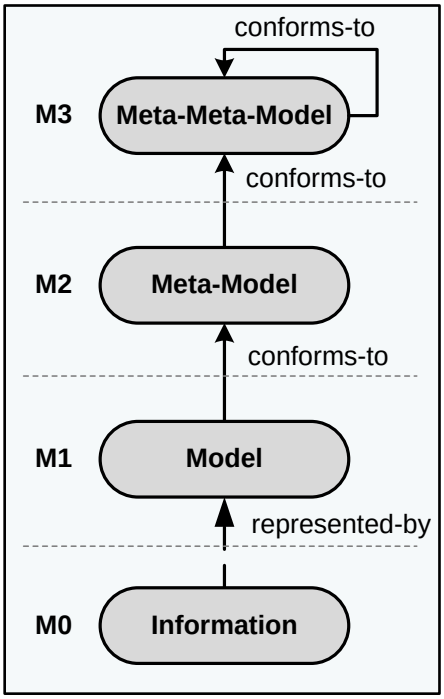


Fig. 2.26 MOF layers - Meta-data Architecture proposed by OMG [162]

meta-meta-model has to conform to itself. The MOF standard defines a meta-modelling language called MOF2.0 [162], to which for instance the BPMN meta-model conforms to. Another more or less aligned variant of a meta-meta-model is the *Ecore* model, used and defined in the Eclipse Modeling Framework (EMF) [225]. Throughout this thesis, MDE principles are implemented using the EMF toolkit/framework.

2.6.3 Methods and Techniques in Model-Driven Engineering

Frameworks for MDE provide methods to create, test, and manipulate models, i.e. manage models as development artefacts. Similarly methods and techniques have been established to support MDE during run-time, e.g. model transformation, model merging, model metrication (establishing measures on models), and model verification, etc. Bézivin et al. distinguishes in [15] between *"modeling in the small and modeling in the large"*. This view is adopted from the difference of *"programming in the small and programming in the large"* [51] established by DeRemer and Kron in 1975. The following two operations belong to the scope of modelling in the small [15]: *model transformation* and *model weaving* - both are further discussed in the following paragraphs. On the other hand, modelling in the large, which is also called *Megamodelling*, is mainly based on model management and an holistic approach to system engineering. Following the paradigm of "everything is a model" a megamodel describes a global view on all involved artefacts in the form of a model [15].

Model Transformation is one of the most important operations in MDE and basically represents the fundamental functionality of the software. It is used for translation from one language into another or to annotate, merge, refine information of two or more source models. The transformation from a number of source models into a number of target models is also called *Model-to-Model Transformation* (M2M) [71]. A M2M transformation generates a target model from a source model and is executed according to a specified model transformation. Adhering to the basic MDE principle the transformation can be considered a model itself. This common pattern for model transformations is referred to as *model transformation pattern* [5]. These transformations can be realised in a number of different ways, e.g. as script, programming code, or through a specialised M2M transformation language such as *Query/View/Transformation* (QVT) [163] or *ATLAS Transformation Language (ATL)* [112, 113]. As a consequence of the "transformation is a model" concept, higher-level transformations are possible, i.e. transformations taking transformations as input and/or generating transformations as output [14].

Generally, there are various applications for M2M transformations. One is to transform a source model, e.g. a business process model, together with additional data, such as process performance related information into one target model that conforms to the same language as the source model. Now, the target model expresses the source model plus the additional information. This example application is called *refinement* [69]. Another special type of transformation is the so called *Model-to-Text Transformation* (M2T), in which a model is transformed into a text. If the resulting text is defined with a metamodel, M2T is classified as a subclass of M2M [71]. M2T responds to the need that models are normally stored based on a concrete syntax, e.g. translating a model conforming to a specific *XML* format.

Model Weaving encapsulates the different kinds of semantic relationships between models and uses these for various activities [48]. Two of the most prominent and relevant examples of semantic relationships are:

- *Annotation models*, i.e. elements of the annotation model have explicit and unique references to elements of another model. E.g., for the online order business process model from Figure 2.2 an annotation model with process performance data can exist that contains performance information for each activity plus links to the respective activities of the original BP process model.

- *Traceability links* between source and target models of an M2M transformation help to maintain a connection between elements of source and target models [111]. Traceability links are usually by-products of M2M transformations but are difficult to create (in an automatic fashion) if the transformations go beyond simple mapping.

2.7 Models at Run-time

Models@run.time (MRT) is relatively new research area that attempts to apply and adopt concepts of model-driven engineering for usage at run-time in order to cope with ever-more dynamically changing software and its environment. Similar to the design-time focused view on models in MDE, in MRT models also represent central software artefacts - but during run-time. In traditional MDE, software development and software execution are separated from each other while modifications on the running system is done in an offline fashion, e.g. via redeployment of changed and compiled software artefacts (*cold swap*). Since modern applications have to adapt quickly to changes in requirements and execution environments during run-time, the distinction between development and execution continues to fade [78]. The initial idea of MRT was to reuse model artefacts from the development phase at run-time to overcome flexibility limitations of the traditional MDE approach [9, 16]. During execution, information monitored from the system enables run-time reasoning to eventually initiate corrective actions at the model level if necessary [227]. As a result techniques in MRT aim to shorten or automate (parts of) the adaptation lifecycle on several accounts: monitoring, reasoning, and modification during execution. The achieved run-time flexibility of self-adaptation, however, requires additional modelling effort since cause and reaction need to be specified which may involve extreme modelling efforts for complex systems. This issue can partly be mitigated by using multiple models capturing different system concerns as opposed to one model capturing the entire system [16, 267].

Blair et al. define a model at run-time as *"... a causally connected self-representation of the associated system that emphasises the structure, behaviour, or goals of the system from a problem space perspective"* [16]. An alternative definition with a greater emphasis on model manipulation is provided in [10]: *"models@run.time is an abstraction of a running system that is being manipulated at runtime for a specific purpose"*. It is further specified that (1) the main kinds of manipulation are *model observation* and *model modification*, and (2) the specific purposes mostly focus on adaptation, monitoring, trace of execution, or debugging [10]. Figure 2.27 illustrates the concept of models@run.time in comparison to the traditional view of models as design artefacts in the MDE approach [9]. In contrast to design-time models which are transformed or compiled to an executable system and thus are separated from the run-time system, MRTs feature a closer link to the system with the purpose of ensuring a *"...good performance during execution"* [9]. This is highlighted in Figure 2.27 by the different distances between model and system for MDE and MRT respectively. The definition and principal is very similar to *reflection* where the system is able to query and manipulate its own structure and state, i.e. is a causally connected self-representation [137]. However, the difference is that MRTs generally operate on a higher level of abstraction (solution space) than reflection (problem space) and as a result allow for a utilisation of model-driven techniques at run-time [9, 16].

In the following different (re-)occurring aspects of existing MRT approaches are dis-

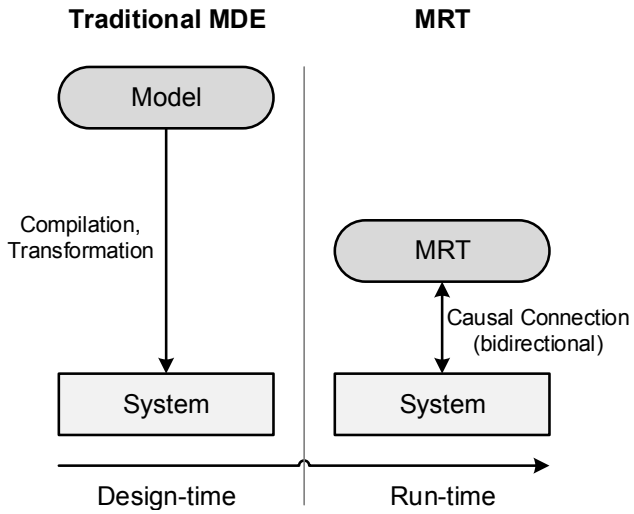


Fig. 2.27 Design Models vs. Run-time Models [9]

cussed, namely their objectives, techniques, architectures, megamodels, and a generalised view on properties of run-time models. A particular focus throughout this discussion will be on aspects of descriptive models at run-time, which is also the subject of the concluding part of this section.

2.7.1 Objectives

Purpose and objectives of MRT can be very diverse. While early approaches mostly focussed on simple monitoring and self-adaptation approaches, e.g. [75, 76], in recent years MRT approaches of a wider application range that are more goal-oriented and user-centric have been proposed [11]. In the following observed classes of objectives of recent MRT approaches are listed [227, 264]:

Monitoring: An essential part of MRT is monitoring the actual state of the system which is the basis for immediate reasoning on the monitored state, e.g. via constraint checking, simulation, and prediction. Monitoring using MRT provides information that helps to trace the applications behaviour and allows for an analysis on the model level, i.e. on a higher level of abstraction and closer to the problem space [16, 227]. Example MRT approaches that include the objective of monitoring of the system are, for instance: (1) solutions based on the Supporting Models @ RunTime (SM@RT) tool to maintain the causal connection between system and architecture model through bidirectional QVT-transformations, e.g. [100, 222, 223], (2) solutions based on the *Triple Graph Grammar* (TGG) approach for architectural monitoring, e.g. [82, 262, 265, 266], (3) Object Constraint Language (OCL)-based monitoring, e.g. [93, 94], (4) using i Nets to recognise [208] or monitor [36] run-time events, or (5) solutions based on the analysis of event log [19] or execution trace [141, 141].

Prediction and Simulation: Based on monitored data immediate and automated reasoning on the current system state (and possibly other alternative states) is possible. Many MRT approaches employ prediction and simulation methodologies as an enabler for adaptive reasoning and/or adaptation decision support. One example MRT approach that is based on simulation is provided by Beltrame et al. which enables effective simulation and debugging through model switching [7]. MRT approaches with prediction-based objectives are often based on transformations to formal model representations and their subsequent analysis (either trough simulation or analytical approaches). Examples are (1) *Queueing Networks* and *Markov Chains*, e.g. built from dynamically created models through system monitoring [31, 80, 152], (2) *(Coloured) Petri Nets*, e.g. to predict the performance of component-based systems [157], or (3) *Bayesian Networks*, e.g. to predict future performance, reliability, and possible violations based on run-time data [61]. However, in some MRT approaches the usage of domain-specific models for simulation and predictions is employed, e.g. to predict the resource consumption of executable components in embedded systems resource models, behaviour models and scenario specifications are utilised in [156].

Adaptation: One of the main objectives of MRT is adaptation during run-time. This can either be facilitated by an operator or in an automated fashion, i.e. as self-adaptation. Often MRT approaches supporting run-time adaptation combine system modification mechanisms with monitoring mechanisms, i.e. establishing a causal connection between models and system in a bidirectional way, e.g. [31, 77, 105, 222, 223]. Furthermore, some MRT approaches utilise prediction and/or simulation mechanisms as enabler for certain types of adaptive reasoning [116, 174]. Adaptation is an objective in the context of many different scenarios such as (1) *dynamic user interface interaction*, i.e. learning from previous user interactions and improving the user experience through continuous adaptation of an interpretable or executable interaction model, e.g. [18, 77], (2) *changes of requirements or in the operational environment*, e.g. through feature models that help decrease the amount of possible system configurations [105] or the usage of the requirements language *RELAX* for self-adaptive systems [283], *change impact analysis*, e.g. through a combination of performance and history-based models to analyse the potential impact of adaptations [160, 174], or *enforcement of Quality-of-Service (QoS)*, e.g. via monitoring data of the execution system and checking the performance model for violations of non-functional requirements [31]. According to [227] non-functional requirements which can be satisfied through MRT-based system adaptation are, for instance, *performance, reliability, efficiency, effectiveness, security, interoperability*, and *usability*. Since this is one of the main objectives, many other domain-specific solutions for MRT-based self-adaptation exist, however, their extensive review is outside of the scope of this thesis.

Other objectives: Several MRT approaches also attempt to meet a set of other objectives. For instance, objectives that originate from advantages of using of models closer to

the problem space (i.e. domain-specific models) such as *abstraction* and *platform independence*, e.g. [82, 262, 265, 266]; According to Szvetits and Zdun in [227] other objectives of MRT approaches include checking and/or enforcement of *consistency*, *conformance*, and/or *policies* as well as *error handling*.

2.7.2 Techniques

MRT approaches apply different techniques to achieve their objectives while at the same time accommodate to the system's setup and specifics. To structure the techniques, they have been split up into three different groups: (1) *system modification*, i.e. maintaining the causal connection from model(s) to system, (2) *system monitoring*, i.e. maintaining the causal connection from system to the model(s), and (3) *general techniques*, i.e. techniques that are not directly attributed to system modification or monitoring but have an important role in managing and supervising MRT systems.

System modification: In order to maintain the causal connection from model(s) to system different techniques have been proposed. One is the usage of *executable models*, i.e. models comply to a language that has an operational semantic and are, much like machine code, directly interpreted/executed by the system. For this model and program share the same representation and are inherently on the same level of abstraction [85]. One example promoting the usage of executable models is *MEMO* [67], a DSML for multi-perspective enterprise modelling that is syntactically and semantically specified using the meta-modelling language MML [66, 67]. MML allows the definition of *intrinsic features*, i.e. enabling the modelling of instantiation as meta-concept [66]. Johanndeiter et al. conceptually propose in [110] the usage of OrgML (a specialisation of MEMO) to be used as executable business process model. Another example is the employment of executable models for Human-Computer Interaction as proposed in [18]. While the usage of executable models simplifies system modifications since changes in the model directly change the system behaviour, it does contradict the abstraction principle of MDE and DSMLs to some extent because problem and solution space are on the same abstraction level. Another technique used for system modification in MRT approaches is via model transformations such as M2M- and M2T-transformations as discussed in Section 2.6.3. MRT approaches using transformations to modify the system are, for instance, solutions based on the Triple Graph Grammar (TGG), e.g. [82, 262, 265, 266], or on bidirectional QVT-transformations, e.g. [222, 223]. On a more general note, a special challenge model-based system modifications need to address is how run-time changes affect current instances of the executable model entities, i.e. modification policies.

System monitoring: In the executable models system monitoring is usually not necessary since the models are directly involved in the system execution and can be queried via reflective methods (e.g. intrinsic features [66, 67]). Transformations may also be involved

in the monitoring of the system, e.g. see TGG approaches [82, 265, 266]. Generally, the technique of extracting state information from the system is called *introspection* [227]. Based on how the state information is extracted from the system three different types can be distinguished in MRT literature:

- *Event log analysis*: This monitoring technique is based on analysing a log containing information about state transitions of the observed system. These event logs have to conform to a specified language in order to represent or be processed into run-time models. One system monitoring approach based on such transition logs is proposed by Mao in [141]: Here traces represent a model-based reflection that provides a "unique visibility" into a system's run-time state and thus enables model-based dynamic analyses. It is furthermore demonstrated that the trace itself can be viewed as a run-time model according to a definition by France and Rumpe in [65] if it conforms to a modelling language. It can certainly be argued that such a trace/log model fulfils both, the abstraction and causal connection requirements usually associated with run-time models. Another approach presented in [19] extracts information from an event log to subsequently check its consistency with the corresponding model, i.e. conformance checking (see general techniques). In [215] events omitted by a web service application are processed "online"[13] and checked for run-time violations which in turn may initiate the enactment of defined recovery plans. Also in [99, 151] direct event processing via CEP or sockets is proposed to extract the system's context.

- *Enhance observed system*: If state transition data is not readily available an alternative approach is to insert monitoring functionality into the observed system. The weaving of data extraction code into the system application is often realised through aspect-oriented concepts, e.g. [151, 271]. Some approaches monitor the system state on the bytecode level, e.g. [93], others through annotations or manipulations on a higher abstraction level such as (1) Abstract Syntax Tree (AST), e.g. [158], or (2) process model, e.g. [101], which then automatically translates into monitoring functionality at enactment. The connection between implementation and model in these cases is often maintained with the help of a traceability model (see Section 2.6.3).

- *Reflection API*: In this case an interface on the target system already exists which allows extraction of state information at run-time. This is achieved with reflection APIs, e.g. provided by reflective middleware solutions such as OpenORB [17] or ReMMoC [86], the reflective data structure H-graph [85], or the multi-perspective enterprise modelling framework MEMO/MML [66, 67].

[13] instead of an event log, a stream of events is directly accessed and analysed event-by-event (see Complex Event Processing (CEP) in Section 2.2.3)

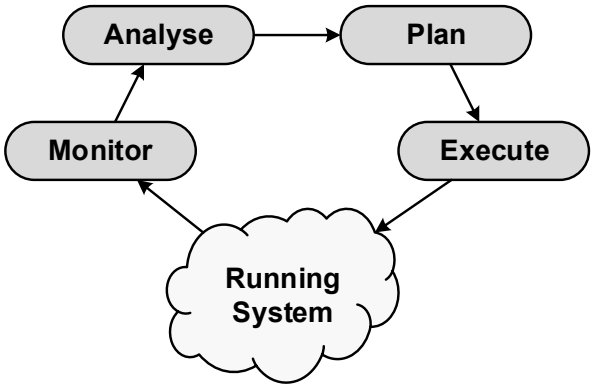


Fig. 2.28 Autonomic Control Loop: Monitor-Analyse-Decide-Execute (MAPE) Mechanism [116]

General techniques: Apart from mechanisms for system modification or monitoring, MRT approaches also use a set of other techniques in order to tackle the objectives introduced in the last section. One of the key concepts of adapting systems at run-time is a strategy that can be described as *autonomic control loop* [116, 174, 271]. One implementation of the autonomic control loop is the Monitor-Analyse-Decide-Execute (MAPE - or MAPE-K with a shared knowledge component) procedure as displayed in Figure 2.28 [116]: In a first step the system (or a part of it) is monitored (see techniques for system monitoring). This data is further analysed, e.g. prediction, constraint violation, etc., and the results are the basis for deciding/planning whether and how the system needs to adapt. In a last step the proposed adaptation is then enacted (see techniques for system modification). Depending on the MRT approach and its objectives the adaptations can encompass structural changes, e.g. in [266], as well as parameter adaptations, e.g. in [61], or both [30]. There are different ways to carry out the adaptive reasoning in the planning phase. Examples are (1) *logic-based reasoning*, i.e. finding a configuration that is satisfying the current contextual constraints and requirements, e.g. [62, 271], or (2) *optimisation-based reasoning*, i.e. configurations are analysed for their quantifiable performance in order to find the best configuration according to specified goals, e.g. [61, 156, 157]. Approaches of both types often employ simulation or other predictive analyses to determine the performance potential of possible configurations or to verify impacts of planned changes, e.g. [61, 156, 157, 271]. According to Fleurey et al. adaptation reasoning requires the following types of input [62]: *context*, *variants*, *constraints*, and *rules*. The output of the reasoning framework is an adaptation that matches the rules based on possible variants as well as context and satisfies the dependency constraints. Vogel et al. additionally include *strategies* and *goals* to allow for a wider range of adaptive reasoning techniques [267]. Following the principle of "everything is a model" Vogel et al. propose in [263] the ExecUtable Run-timE MegAmodel (EUREMA), a domain-specific modelling language and a runtime interpreter for adaptation engines, i.e. a megamodel language to model and execute autonomic control loop techniques.

Whereas the autonomic control loop can be regarded as a kind of mega-modelling

technique, i.e. corresponds to "modelling in the large", other techniques used by MRT approaches rather correspond to "modelling in the small". Such smaller-scale mechanisms that have not yet been discussed are, for instance, (1) *model conformance*, i.e. a technique that usually compares information of different language levels (see Figure 2.26): whether a model conforms to its meta-model or if model instance information is consistent with the corresponding model, (2) *model comparison*, i.e. an operation that compares two models and provides information about the differences, or (3) *model annotation*, a specialisation of model weaving (see Section 2.6.3). For instance, model comparison is often used to compare current with designed model to identify deviations or with a planned model in order to determine which system modifications have to be carried out [62, 99, 267, 271].

2.7.3 Architectures and Megamodels

A differing perspective on recent MRT approaches is to investigate where the run-time models reside in relation to the system and agents. For this two different dimensions are considered: (1) an architecture-based view and (2) a view describing the interactions between system, run-time models, and agents, i.e. information flows expressed via megamodels [15] (see Section 2.6.3).

Architectures

Dealing with models at run-time requires architectures that support (as a minimum) introspective and/or system modification capabilities, i.e. provide an answer to the question where the models reside and through which channels an information exchange between models and system can be achieved. Architectural differences are mostly related to model access, virtual location of the models, or the level of supported dynamics, e.g. adaptation, simulation, optimisation [227].

Szvetits and Zdun identified in their review of MRT approaches the following five architecture types [227]: Monolithic, local dataflow, model-aware middleware, communication middleware, and repository architectures. MRT approaches using a *monolithic* architecture encompass MRT and system functionality in one single component without separation of concerns, e.g. [158]. In contrast, the *local dataflow* architecture separate concerns into local components which communicate with inter-process or in-process mechanisms like sockets and pipes, e.g. [99]. A third (and very prominent) approach in the MRT domain comprise middleware architectures which use dedicated and distributed components for additional abstracted functionality like controlling, monitoring, or communication, e.g. *model-aware middleware* [31, 80, 208, 222, 262, 265, 266] or *communication middleware* [80, 101, 208]. Yet another approach is that of a *repository* architecture which allows for concurrent access operations (e.g. publish, subscribe, add/delete, update, etc.) and archiving for models via a central repository component, e.g. [30]. Note, that this differentiation is not exclusive, i.e. approaches can use more than

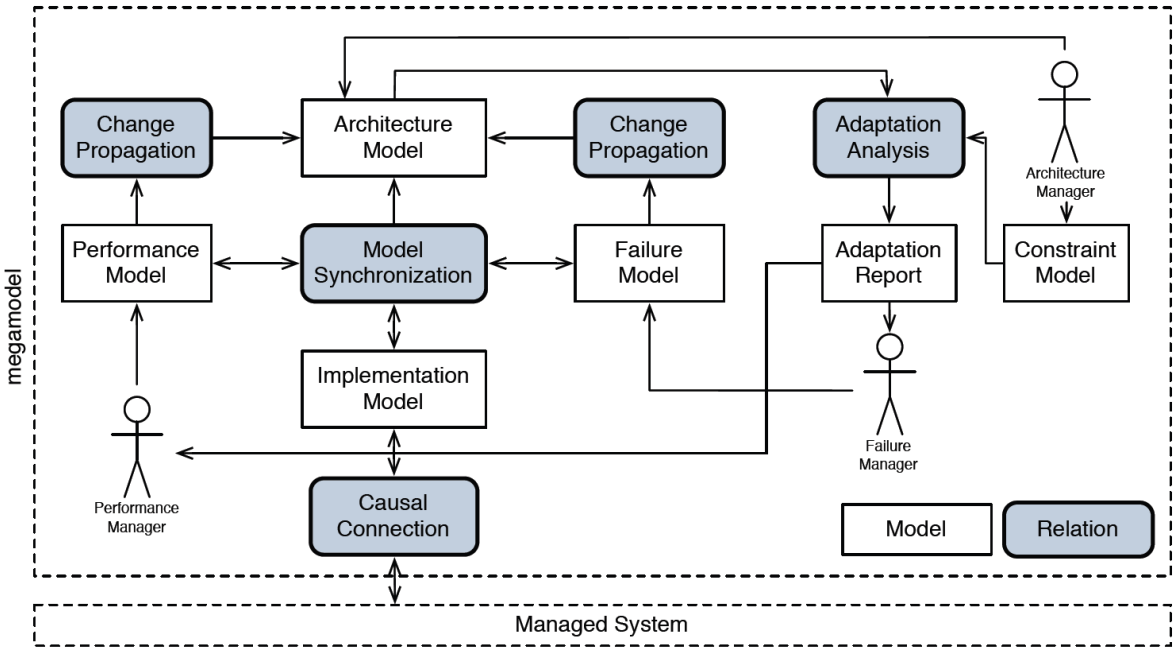


Fig. 2.29 Megamodel Example for Self-adaptive Software [267]

one of these architectures, e.g. model-aware and communication middleware architecture in [80, 208].

Megamodels

Another perspective on where run-time models reside is through the megamodelling view, i.e. modelled relations between the system and run-time models, agents, and operators of different purpose. Vogel et al. carried out an initial investigation of megamodelling for MRT and proposed reference megamodels for different MRT scenarios in [267].

One of the scenarios investigated is that of self-adaptive software which employs several runtime models simultaneously for monitoring and adapting as proposed in [262, 265] (see Figure 2.29). The main feature of this scenario is that a run-time *Implementation Model* resides in the MRT supervision component and conforms to a solution-space representation, i.e. is complex, platform-specific, and at a low level of abstraction [267]. Abstractions on top of this solution-space model to specific problem-space models, i.e. *Architecture*, *Failure*, and *Performance Model*, is achieved via incremental and bidirectional *Model Synchronisation* techniques. On the other hand the *Causal Connection* between model and system is in this case achieved via reflective techniques since the *Implementation Model* is a true self-representation. This scenario thus represents reflective use cases close to the solution space.

A second representative scenario investigated in [267] is that of model-driven configuration management system for IT service management as originally proposed in [81] (see Figure 2.30). Here the main feature is that a differentiation between a monitored *As-Is-Configuration Model* and a reasoned *To-Be-Configuration Model* is proposed. While this approach supports the notion of a run-time model on a higher abstraction level (i.e. a

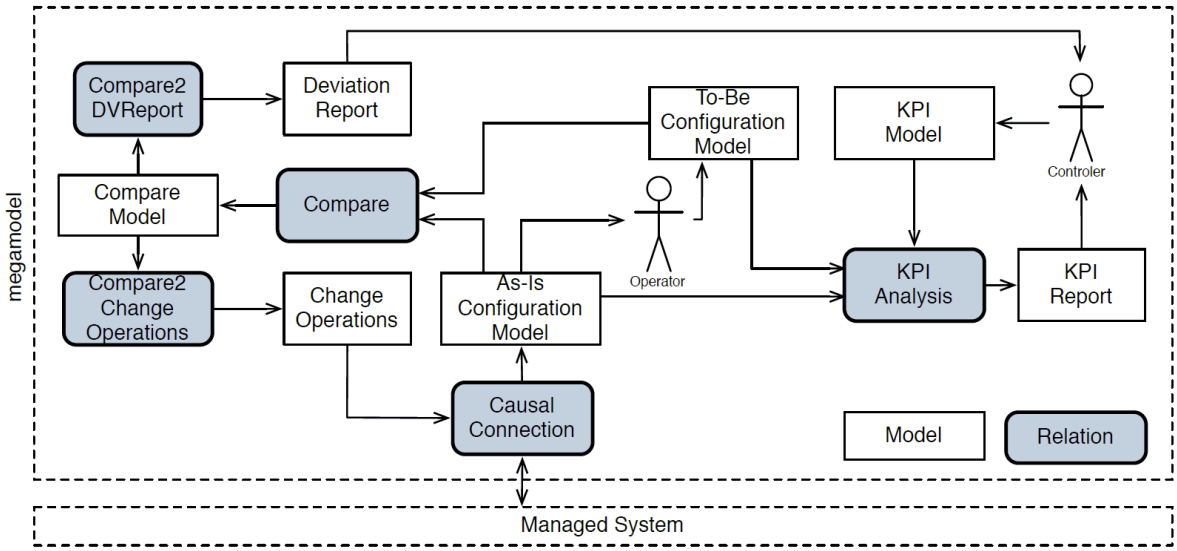


Fig. 2.30 Simplified MRT Megamodel Example for IT Service Management [267]

Configuration Model), stronger requirements to maintain the causal connection between system and the as-is/to-be models have to be met. This scenario and the proposed separation between as-is and to-be model can be seen as reference for use cases that are concerned with run-time models of a high abstraction level (i.e. located in the problem space) and thus fits better in the domain of run-time business process models.

2.7.4 Generalising Models at Run-time

Run-time models are used in different domains and serve different purposes, i.e. they are typically problem oriented. Examples of domains and abstractions of run-time models are Abstract Syntax Tree [158], Petri Net [36, 208], architecture models [82, 265, 266], state machines [99, 129], and many more. Generally, the model's properties are dependent on its domain and purpose. Still, similarities can be found that are more or less existent in most of the run-time models.

One approach of classifying model elements of MRT is presented in [25] in which an analysis of model dynamics and executability has been carried out. Therein the following classification of elements of *executable* run-time models has been identified:

- Definition part: the static part of the model which is defined at design-time
- Situation part: describing the dynamic state of a system during execution
- Execution part: specifying the transitions from one state to another

Because of the classification's focus on executable models it does not fully apply to *general* run-time models [129], i.e. not every run-time model is an executable model. For instance, run-time models with the purpose of monitoring do not necessarily have to have a definition part; some are built completely at run-time (e.g. by data mining algorithms). The inapplicability for general run-time models of this element classification motivated

Lehmann et al. [129] to focus on classifying run-time model elements based on the causal connections of the model. The causal connections in a MRT are either of a descriptive or prescriptive nature [212]:

- A model is descriptive if all statements made in the model are true for the System Under Study (SUS), i.e. every relevant change of the system is captured in the descriptive part of a run-time model.
- A specific SUS is considered valid relative to a prescriptive model if no statement in the model is false for the SUS, i.e. the space of possible system states is defined by the prescriptive part of a run-time model.

In general, the specification ratios of descriptive and prescriptive parts in a run-time model differ dependent on its purpose. That is, a MRT that focuses, for instance, on monitoring has a strong focus on descriptive parts (e.g. [141, 208, 226]) and a MRT that focuses on executability has a dominating prescriptive role (e.g. [153]). In addition to the prescriptive and descriptive parts of the model, Lehmann at. al identified that valid model modifications for both, descriptive and prescriptive, and the actual information flow of the causal connection are part of a general run-time model, too. The resulting classification to define elements of meta-models for general run-time models is the following [129]:

- prescriptive part - how the model should be
- descriptive part - state of the SUS at run-time
- valid modifications of descriptive part during run-time
- valid modifications of prescriptive part during run-time
- causal connections - modelling the information flow between the model and its SUS

The classification of elements for run-time models by Lehmann et al. [129] is shown in an example in Figure 2.31. Assuming the system has only a finite number of states then the prescriptive part would reflect all these states and the descriptive part would consist of the single state the system is in at the moment. The valid modifications of the descriptive part would determine the transition from one state to another, it represents the execution logic of the system. Additionally, through the notion of modifications of the prescriptive part the run-time model would be available from within the run-time model itself, i.e. be self-representative. Models that have the properties of self-representation and causal connection are called reflective [38]. As elaborated in earlier sections, a run-time model does not necessarily has to have the property of self-representation, e.g. monitoring models are causally connected to the system sufficient for their purpose without having the ability to change the system. Also, the ratio of prescriptive and descriptive parts are dependent on the purpose of the model: For instance, prescriptive parts of a monitoring MRT can be descriptive in a MRT for dynamic structural adaptation.

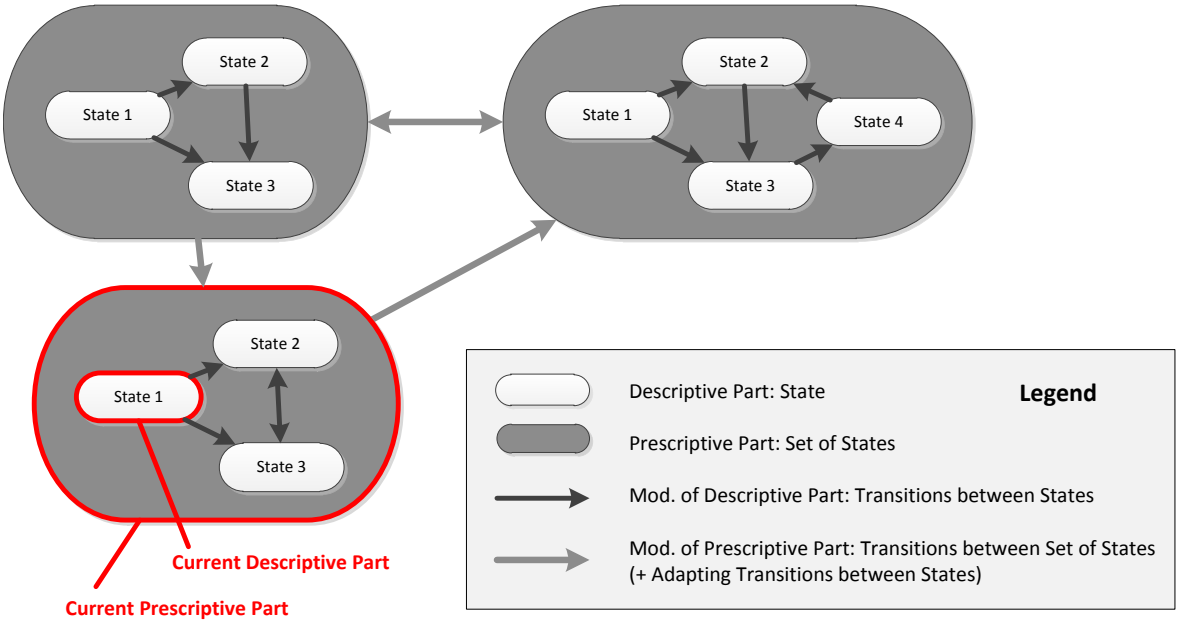


Fig. 2.31 Descriptive vs. Prescriptive Parts

There is, however, one general issue that makes this classification only partly suitable for a general MRT: The classification captures the self-representation property only partly because valid modifications for the descriptive parts should be able to change at run-time as well in order to support full self-representation. Assuming a state is added to the prescriptive part of the model then transitions describing how to reach this state would also have to be defined (see Figure 2.31), i.e. add valid modifications of the descriptive part. It can be argued that the logical adaptation of the classification to overcome this issue is to declare the valid modifications of the descriptive part to be a part of the prescriptive part of the model. A good example of this fact are business processes models: They are in their original sense mostly prescriptive but also already define a workflow, i.e. the valid modifications/state transitions of the system.

2.7.5 Descriptive Models at Run-time

In many cases, approaches in the MRT domain are regarded as holistic solutions for self-adaptation for which both directions of the causal connection between a run-time model and system are to be maintained. This is often achieved either (1) through a shared representation in the form of an executable model on the solution space (see Figure 2.29 and Section 2.7.2), e.g. [18, 110, 153]), or (2) through bidirectional transformations (which allow for some level of abstraction to the problem space), e.g. [82, 222, 223, 265, 266].

However, in domains where a greater gap in abstraction between models and managed system prevails, it is more difficult to maintain a causal connection between system and the run-time models. One such example domain is "IT Service Management" as introduced in Section 2.7.3 (see Figure 2.30 on page 77): In this scenario it is differentiated between an "as-is" and a "to-be" model; while the "as-is" model contains

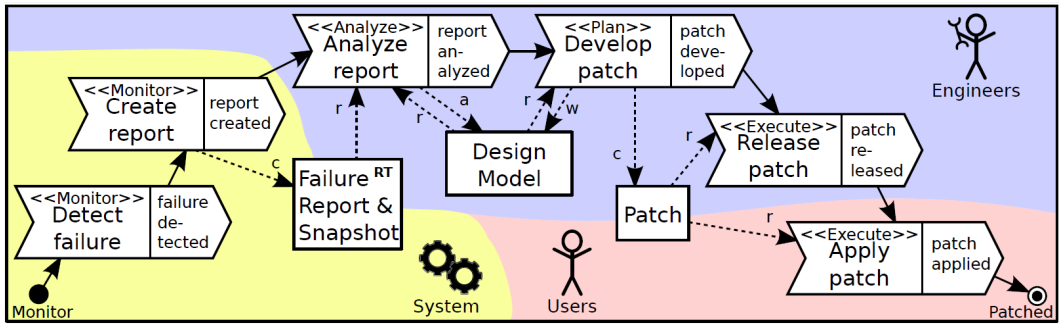


Fig. 2.32 Run-time Models for Monitoring in Maintenance [264]

monitored/extracted information of the current state of the managed system, the "to-be" model specifies what the system should behave like [267]. Using the terminology established in the previous section, that means the "as-is" run-time model consists solely of a descriptive part and thus may also be called a *descriptive run-time model*. Similarly, the "to-be" model solely consists of a prescriptive part and thus can be also called a *prescriptive run-time model*. The separation of concerns through a differentiation between descriptive and prescriptive run-time models has the additional benefit of allowing for other actors than the MRT system to be the driver of change, thus supporting the management of so called *open systems* [79, 80]. That means, this concept allows for dealing with a system change caused by an external source (e.g. exceptional behaviour, changing environment). In contrast, holistic closed-loop concepts based, for instance, on executable models or bidirectional transformations assume an isolation of the system from external influences, i.e. the system will always act according to the behaviour specified in the model and not allow for deviations.

Another benefit of this differentiation is showcased by Vogel and Giese in [264] where different maintenance and self-adaptation scenarios for the adjustment of a faulty application (original approach proposed in [35]) are investigated. For the different scenarios all actors and models are associated to either the system, the user, or the engineer, i.e. defining the managing entity of the agents and models. The association to the entities for each scenario is dependent on the purpose of the MRT solution but also on what kind of interactions with the system are permitted, i.e. what model access possibilities are supported by the managed system. Note, that the entities basically represent different stages of an application's traditional lifecycle, i.e. engineer → design, user → implementation, system → enactment/execution. Figure 2.32 shows the associations for the scenario of "monitoring in maintenance". With increasing autonomy of the investigated scenarios in [264], e.g. "execution in maintenance" and "self-adaptation", an increasing number of models and agents move into the system's sphere of influence, i.e. become run-time models/agents. The different scenarios showcase that differentiating between descriptive and prescriptive run-time models is helpful to (1) separate concerns, i.e focus on particular challenges for system monitoring, adaptive reasoning, and system modification individually, and (2) scale the level of autonomy to an appropriate level, i.e. appropriate for the

abstraction level of the domain (e.g. autonomous reasoning possible vs. domain-expert reasoning required) while taking the available support for system access into account (e.g. autonomous system modification possible vs. software-expert required for change implementation). In conclusion, the additional conceptual flexibility achieved by the differentiation of descriptive and prescriptive run-time models yields the following important benefits in comparison to holistic closed-loop concepts:

- allows for other drivers of change than the MRT supervision system,
- enables a greater variety of objectives, e.g. conformance checking, error handling,
- different scales of lifecycle autonomy are supported, e.g. self-adaptation vs. adaptation reasoning by domain experts, and
- separating concerns for system monitoring, adaptive reasoning, and system adaptation.

All four benefits are helpful when addressing domains that feature models of high abstraction levels (e.g. business processes) where the causal connection between model and system is difficult to establish. This is further emphasised by the shift of focus in MRT publications in recent years: They concentrate less on low level self-adaptation issues but are more specialised and cover a wider application range, i.e. are user-centric, domain-specific, and goal-oriented [11].

In the context of the differentiation between these two types of run-time models and the consequential separation between the two directions of the causal connection, a descriptive run-time model can be defined as *a causal reflection of the associated system's current status from a problem space perspective*. Research in the domain of descriptive run-time models can therefore focus on domain-specific challenges that are concerned with reflecting different dimensions of change, e.g. reflecting change on both levels: model type and model instance, and maintaining a causal connection from system to descriptive run-time model, e.g. overcoming a possibly existing abstraction gap between the available system data and the domain-specific model. Figure 2.33 demonstrates the relation of a descriptive MRT to the system as well as to potential other relevant types of models that are directly interacting with the system, i.e. design model and prescriptive MRT. Models on the meta level which are not directly interacting with the system, e.g. models for reasoning such as rule, constraint, or goal models [62, 267], are not regarded in this context since they address a different concern (that of adaptive reasoning). The figure enhances the comparison between MDE and MRT concepts from [9] shown in Figure 2.27 on page 70. Similarly it highlights that run-time models (descriptive and prescriptive) have a conceptually closer link to the system than design model. In this simplified view three different scenarios for descriptive MRT relationships are displayed:

1. In the first scenario (all elements in red, purple, or black colours) a design-time model is the basis for establishing a system instance via compilation, transformation, implementation, or configuration depending on the employed MRT concept

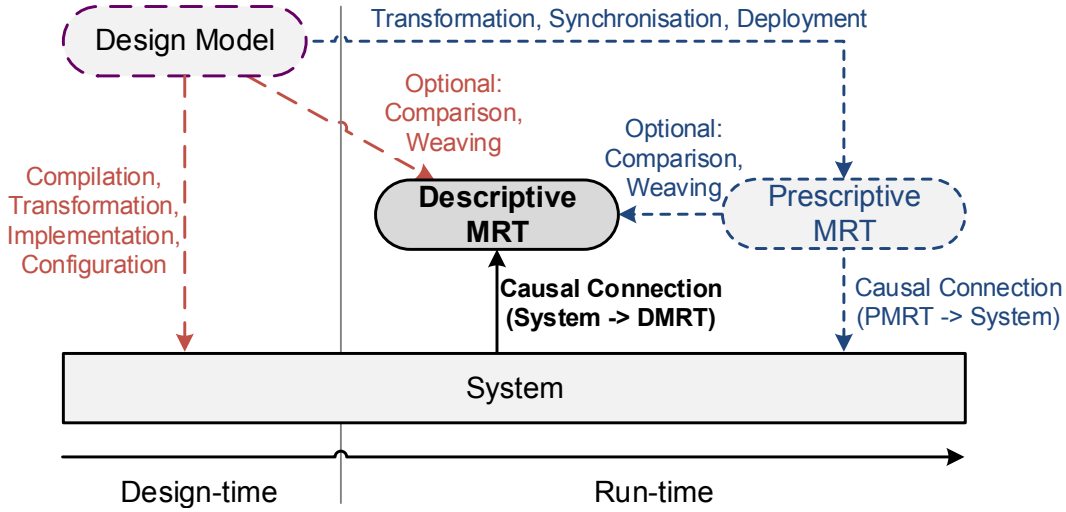


Fig. 2.33 Schematic View on Relations between Descriptive Run-time Models, other Model Types, and the System

of the scenario. The descriptive MRT is conceptually linked with the design model via model weaving techniques, e.g. model enhancement, or model comparison for reasoning purposes, e.g. identifying constraint violations based on monitored run-time data [61, 93, 94, 99, 215].

2. In the second scenario (blue, purple, and black colours) a prescriptive MRT is defining the functionality of the system instance via system modification techniques (see Section 2.7.2). The prescriptive MRT is usually connected with a design model and updated via transformation, synchronisation, or deployment depending on the employed MRT concept of the scenario. Similar to the scenario 1 the descriptive MRT is conceptually linked with the prescriptive MRT via model weaving techniques or model comparison. An example of this scenario is presented in [81, 267] and shown in Figure 2.30 on page 77 with "As-Is-Configuration Model" being the descriptive MRT and the "To-Be-Configuration Model" the prescriptive MRT. Other examples of this scenario based on a prescriptive and descriptive MRT relationship are proposed in [62, 271].

3. In the third scenario (only black colour) the descriptive MRT is a single entity independent from possibly existing design models or prescriptive MRTs. It has the single purpose of directly and accurately discovering problem level information and/or the system's reflective state from solution level data to enable further reasoning (this may include model comparisons however). Solutions of this scenario of independent descriptive run-time models are proposed for architecture models [208], trace models [141, 142], Queueing Networks [80] or protocol behaviour models [79].

Descriptive MRTs in scenarios 1 and 2 often feature *model weaving* techniques which link to higher level information from the design model or prescriptive MRT. These cases focus on *monitoring* the system's state on the instance level and associate it with the (existing)

information on the model level, e.g. [61, 93, 94, 99, 215]. This is different in scenario 3 (and scenarios 1 and 2 based on model comparison) for which additionally the higher level (problem space) models have to be *discovered* from lower level (solution space) data received from the system, e.g. [79–81, 141, 142, 208, 267]. While this makes it harder to maintain the causal connection between system and the descriptive MRT it allows for more system flexibility (as discussed earlier in this section). In the domain of business processes the method of extracting problem space models (BP models) from solution space data (BP event logs) is referred to as *process discovery* and discussed in Section 2.4.

Note, that the definition of descriptive MRT provided is strongly associated with that of context models [62, 151, 267] or situation models [25], similarly describing the operational environment of the system. Context is defined in [53] as *"...any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."* One difference, however, is that context models can also include environmental context (e.g. the current temperature or customer satisfaction) which is not describing the application system but an entity that may be influencing it. Another difference is that it is often not associated with structural information (on the problem space level) but with quantifiable measures or key-value pairs, e.g. in [62, 151].

In summary, with regards to the introduced objectives (see Section 2.7.1) descriptive MRTs aim to fulfil that of monitoring, abstraction, and platform-independence but also act as enabler for the other objectives such as simulation, prediction, adaptation, and fault error handling. The different techniques for obtaining the information in order to maintain the causal connection are discussed in Section 2.7.2: System Monitoring, e.g. event log analysis, system enhancement, or reflection APIs. The resulting descriptive MRTs are diverse and can range from simple measurable sensor data to discovered structural information on the domain-level. In the former case model weaving techniques are often employed to connect domain-level models and low-level sensor data, e.g. through model enhancement. In the later case elaborate data mining algorithms have to be employed to infer or update the domain-specific descriptive MRTs.

2.8 Summary

In this chapter state of the art literature with regards to the topics of business processes, their management, flexibility, discovery, and performance analysis as well as Model-Driven Engineering (MDE) and Models at Run-time (MRT) was presented and discussed. First, an introduction to BPs, their components, perspectives, and basic terminology was given in Section 2.1. This chapter showed that for each of the individual topics extensive research has been carried out and many approaches and solutions in these individual domains exist. Key take away points of this literature review are:

- Solutions or standards for languages or process analysis disciplines are strongly as-

sociated with a BPM lifecycle step: For instance. *BPMN* is an a-priori language used at *design-time* with the main purpose being documentation and communication between stakeholders, or *Business Activity Monitoring (BAM)* solutions are run-time analyses approaches (mostly focussing on KPI/PPI calculation or violation) developed for the enactment phase. These and other BPM lifecycle associations of languages and process analysis disciplines were identified in Section 2.2.

- Changes to the BP can occur in certain forms and extents (see Section 2.3). Many taxonomies exist classifying model changes from the enactment perspective, i.e. when the change is applied. In the context of descriptive run-time models it is especially important to identify what changes may occur during run-time: momentary change, deviation, permanent change, underspecification through late modelling.

- This chapter also showed that extensive research in the area of Process Discovery has been carried out (see Section 2.4). Process Discovery provides solutions to establish causality from system to BP models, however, the traditional approaches are carried out in an offline/a-posteriori manner. The more specialised fields of concept drift (data mining term for type level changes) and online process discovery techniques have been considerably less researched with only a small number of contributions. Especially the areas of process discovery and online process discovery are relevant to identifying gaps in current literature in addressing the goal of establishing a causal connection from system to BP model at run-time.

- In the literature review for BP performance decision support (see Section 2.5) it was shown that the capturing and prediction of key performance indicators (KPIs) or more domain-specific process performance indicators (PPIs) is an important field in BP research with many solutions published. With regards to the use case of performance prediction two fundamentally different approach types exist: One based on mathematical models and the other based on simulation. It was identified that the simulation approach performs slower but is more accurate in terms of short-term predictions (provided it is based on an accurate model).

- Model-Driven Engineering was introduced as a technique to manage the increasing complexity of modern software platforms, much like Business Process Management Systems, in Section 2.6. The reviewed literature describes the systematic usage of models as development artefacts throughout the software lifecycle.

- An in-depth literature review was carried out in the MDE topic of models at run-time. A special focus throughout this discussion were on solutions in the domain of adaptive systems that feature descriptive models at run-time or contributing to their understanding and application.

The next chapter will be analysing the literature presented here for existing gaps with regards to the thesis' topic of descriptive business process models at run-time.

Chapter 3

Gap Analysis: Descriptive Models at Run-time in Business Process Management

Following the review of state-of-the-art literature in the domains of Business Processes as well as Model-Driven Engineering and Models at Run-time in particular (Chapter 2), this chapter discusses the fusion of these domains. In the context of this fusion the reviewed literature is revisited and it is discussed to what extent the three individual objectives of descriptive models at run-time are addressed and what limitations do still exist. This gap analysis represents the defining input for the subsequent contribution chapters.

Figure 3.1 shows the outline of this chapter and the relation to the remaining contribution chapters. First, in Section 3.1 the feasibility of employing MRT concepts in the domain of business processes is discussed in general. The discovered general findings are relevant for the main goal and all individual objectives of this thesis (see Section 1.3). Hence, the subsequent three subsections discuss the respective limitations of (1) business process models at run-time (Section 3.2), (2) process discovery at run-time (Section 3.3), and (3) process performance prediction at run-time (Section 3.4). The individual findings are each mapped to the respective main contribution as shown in the Figure: (1) Chapter 4: *Specification of Descriptive Business Process Models at Run-time*, (2) Chapter 5: *Establishment of Causal Connection* and (3) Chapter 6: *Reasoning on Descriptive Business Process Models at Run-time*.

3.1 Models at Run-time meets Business Process Management

As discussed in Section 2.2, existing BP standards can be distinguished according to the different phases of the BPM lifecycle (see Figure 2.3 on page 18) they address, e.g. design vs. enactment vs. diagnosis phase, and the purpose they serve, e.g. communication be-

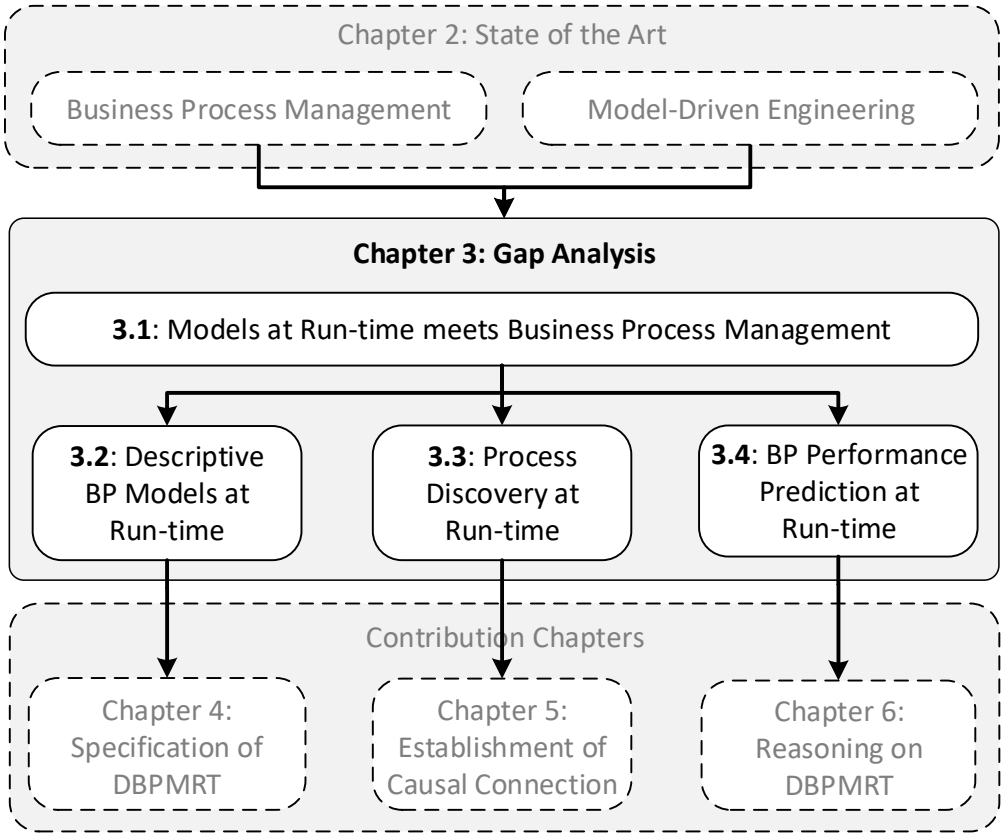


Fig. 3.1 Outline of Chapter 3

tween stakeholders vs. execution vs. analysis. On a higher abstraction level they can be distinguished between a-priori, i.e. before execution, and a-posteriori, i.e. after execution, model standards. In this traditional view both, a-priori and a-posteriori models, are rather disconnected from the running business process management system, i.e. display lack in run-time features. This makes them less suitable for today's modern and volatile organisations which require their BPs to adapt and evolve continuously in order to meet changing demands and constraints inflicted by internal or external sources. To address this, one of the main challenges in BPM is to further automate parts of the lifecycle, i.e. bring the BP-domain models closer to the system (see Section 1.2).

Models@run.time (MRT) which are discussed in Section 2.7 offer techniques and a principled guide to support adaptive systems in a volatile environment. Their main purpose can be described as reducing the distance between domain models and system (see Figure 2.27 on page 70) to make the process of adaptation more automated. Since the domain of business processes is to a large extent a model-based domain (albeit many modelling languages and standards are serving a diverse set of purposes) the assumption is that the consolidation of MRT solutions provides concepts that also help reducing the model-system-distance in the BP-domain, i.e. promoting a closer link between BP models and the BPMS. An additional similarity indicating a benefit of consolidating the BPM and MRT domains can be observed between the BPM lifecycle and the autonomic control loop, e.g. the Monitor-Analyse-Plan-Execute (MAPE) as displayed in Figure 2.28

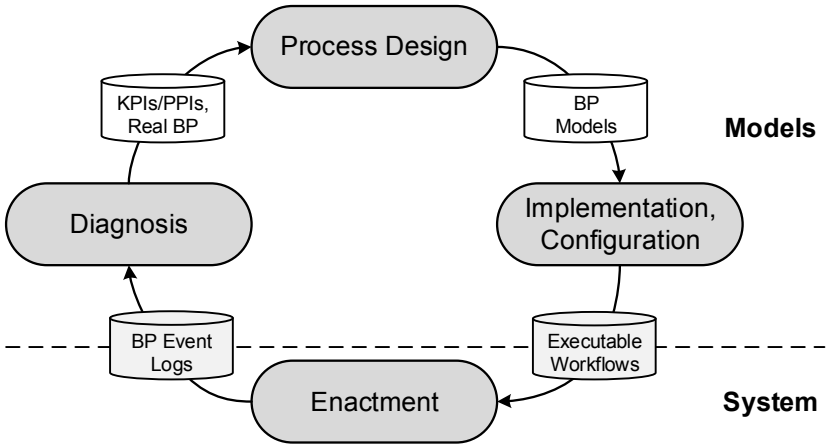


Fig. 3.2 BPM Lifecycle and Information Artefacts Exchanged Between Phases

on page 74: Both, the BPM lifecycle and the MAPE approach, are identical in substance with similar phases representing a control loop. Figure 3.2 shows the BPM lifecycle annotated with the respective information artefacts that serve as interfaces between the different phases of the lifecycle, i.e. BP models, executable workflows (code on the solution space), BP event logs, and PPIs/KPIs or discovered BP models. Ultimately, the interface linking the models and the run-time system in the BP-domain is twofold: (1) *executable workflows* in the form of code artefacts that represent implementations of the planned BP model which enforce the execution of some or all parts of the BP; (2) *BP event logs* that capture fine-granular changes in the transactional state of the system (see Section 2.2.2) which can be analysed and processed into higher-level information, e.g. how business processes are executed in *reality*, i.e. process discovery (see Section 2.4), or measurable values representing certain performance aspects, i.e. KPIs or PPIs (see Section 2.5).

3.1.1 Special Characteristics in Business Process Management

Generally, two main reasons are prevalent in the BP-domain that make the establishment of a causal connection BP models and BPMS as stipulated in MRT approaches very difficult: (1) a high abstraction gap between problem and solution space and (2) uncontrolled deviations from the BP model during the BPM lifecycle phases of implementation and enactment. Both are discussed in more detail in the following.

High Abstraction Gap

Different to the BP-domain, many models in MRT solutions evidently operate on a lower abstraction level (see Section 2.7), e.g. Petri Nets [36, 208], architecture models [82, 265, 266], abstract syntax trees [158], etc. With regards to the BPM lifecycle, there is a considerable abstraction gap between the low level models acting as interface to the system, i.e. the worklfow code artefacts and the event log, on the one hand, and the higher-level BP-domain models on the other hand. While the former dictate (workflow) or describe (BP event log) fine-granule system transitions, the latter are generally associated with strate-

gic goals and serve more abstract purposes like documentation, communication between stakeholders, and conceptual planning (see Section 2.2.1). This gap in abstraction is best demonstrated by the level of human involvement still required in each of the phases of the BPM lifecycle:

1. **Design:** The domain expert responsible for the design of BP models is referred to as business analyst. Her profile includes extensive knowledge about all things concerned with the business process domain but only limited knowledge about the system and actual implementation and enactment concerns (see Figure 2.25 on page 65).

2. **Implementation/Configuration:** The higher the abstraction level of the business process model the more effort is needed to manually or semi-automatically carry out the implementation of a planned process or its adaptation. Common practice (see Section 2.2.1) is that a graphical standard (e.g. BPMN) is used to design the business process model, then an interchange standard (e.g. BPDM) is utilised to transform that into an execution standard (e.g. BPEL). This phase is accompanied with additional configuration or implementation efforts since BPs are domain-specific but not system-specific.

3. **Enactment:** Perhaps different to most other IT-related domains is the human involvement in the enactment phase. Workers are regarded as resources that carry out *human activities* (see Section 2.1). As a consequence the enactment of a BP is usually not fully automated but only computer-aided.

4. **Diagnosis:** The diagnosis phase is the most advanced of the BPM lifecycle in terms of autonomy, i.e. only limited human involvement is required. For instance, BAM solutions exists that allow for the automatic computation of PPIs and KPIs once they have been properly configured (see Sections 2.2.3 and 2.5.1). However, as discussed in Section 2.4.1 the discovery of the actually executed control-flow based on BP events is a multi-goal optimisation and thus a non-trivial and human-guided task, i.e. expert knowledge is required to balance between under-fitting and over-fitting results.

Uncontrolled Deviation

Additional to the abstraction gap, a second important reason for the lack of causality in the phases of the BPM lifecycle is uncontrolled deviation. Real life use cases like those provided by TIMBUS and SAP-internal projects showed that the actual execution often deviates from the designed business process. This is a result of specific characteristics in the BPM-domain and (again) best demonstrated by the high degree of human involvement in two of the BPM lifecycle phases:

1. **Implementation/Configuration:** The BP as designed by the business analyst is an often under-specified and over-simplified abstraction of reality, i.e. the real implementation of a BP is usually different due to system-specific and practical reasons.

2. **Enactment:** As mentioned earlier, a very BP-domain specific characteristic is the human involvement in the enactment phase. A direct result is the lack of enforcement, i.e. the semi-automatic execution of a BP allows for a certain degree of uncontrolled deviation from the designed and/or implemented BP. The core reason is again the lack of practicality of the designed BP. Examples are: (1) some predicted conditions do never in reality apply, (2) exceptional or unforeseen conditions occur that demand ad-hoc adjustments in the execution flow, or (3) a gradual deviation from planned BPs towards a more applicable/optimal (in real life) execution flow takes place.

Both cases indicate that the management and lifecycle of business processes is subject to major or partial influences from outside the controlled system, i.e. uncontrolled deviations from the planned BP occur during implementation and run-time.

3.1.2 Business Process Models at Run-time

Solutions in the MRT domain offer a set of concepts and techniques to deal with complex systems such as BPMSs. However, to maintain the casual connections between system and model as stipulated by MRT some form of automation in the different phases of the BPM lifecycle is required. Hence, many of the MRT principles are difficult or even impossible to adopt since the BP domain has to deal with the two additional concerns that usually do not apply for them: (1) the large abstraction gap between run-time system (solution space) and BP model (problem space) and (2) that the management and lifecycle of business processes are human-guided and subject to external influences.

For instance, many of the MRT solutions provide holistic approaches for an *autonomic control loop* for specific domains (see Section 2.7.2). Approaches adopting an autonomic control loop for the BPM lifecycle can also be found in the domain of BPM. They provide techniques for modelled or ad-hoc adaptations of business processes as presented in Section 2.3 and are based on a shared representation, i.e. the BP model is interpreted by the system (BP model = solution space = problem space). While these holistic solutions for *executable and adaptable workflows* have been an important research topic for almost 20 years, they have not yet been successfully adopted in industry and remain a rather academic topic. It can be argued that this due to legacy issues, e.g. integration problems with SAP systems or other enterprise software. However, in this thesis it is argued that this is also a result of practical reasons concerning the complexity of BPMSs. While holistic BP approaches generally provide solutions to mitigate or address the issue of the abstraction gap (solution space = problem space), they assume major simplifications of which two are listed in the following:

1. they only focus on workflows (which can be likened to automated BP models), and
2. they assume a closed, self-contained system where no other sources can cause a change[1].

Additionally, it can be argued that modelled adaptations as proposed by many adaptive MRT solutions are not easily applicable in the BPM domain due to

1. *redundancy*: since BPs already allow for modelled flexibility (e.g. optional behaviour, choices) a business analyst would rather model known flexibility directly into the business process (a domain she is familiar with) than model them as BP adaptations, and
2. *autonomic enactment*: the enactment of a BPMS modification in an automated way remains a very difficult task since many modification policies cannot simply be applied to all the active BP instances at run-time.

In summary, these holistic solutions do address the issue of the abstraction gap (by simply raising the abstraction level of the solution space), but do not account for the issue of uncontrolled deviations and are often too expensive to model (modification policies, including all eventualities, etc.). This is perhaps the main reason why holistic approaches based on adaptive and interpretable workflow languages which support an autonomic control loop for the BPM domain have only been adopted in domains that employ fully automated BPs in self-contained systems.

The two special characteristics (big abstraction gap and uncontrolled deviation) are also the main motivation for another very prominent research field of BPM: That of developing process discovery algorithms (see Section 2.4.2) to discover the "real" underlying business process. In contrast to the flexible workflow initiatives, process discovery became widely adopted in industry in recent years. This is an indication that the separation between a-priori and a-posteriori models is not only prevalent but also to some extent an inherent requirement given the special characteristics of this domain. So, on one end of the spectrum resides the concept of a-priori and a-posteriori models disconnected from the system which does not meet the requirements of systems in which business processes are highly volatile with possible changes over time. On the other end resides the concept of business process models being handled completely at run-time like with flexible workflows which appears to be a step too far with the observed weaknesses of heavy upfront modelling and no accounting for uncontrolled deviations. As discussed in Section 2.7.5 the challenges of establishing a causal connection between models and system in a complex and high-level domain like business process management which additionally has to deal with uncontrolled deviation are very difficult to address in a holistic fashion.

[1] In such systems all properties of a high-level change are known from the perspective of the change enactment. However, in more liberal BP environments these type-level changes are not recorded in the BP execution event logs and are therefore not available to be analysed. For this reason the detection of these higher-level changes based on low-level event streams is a non-trivial matter and results in new challenges discussed in Section 3.3.

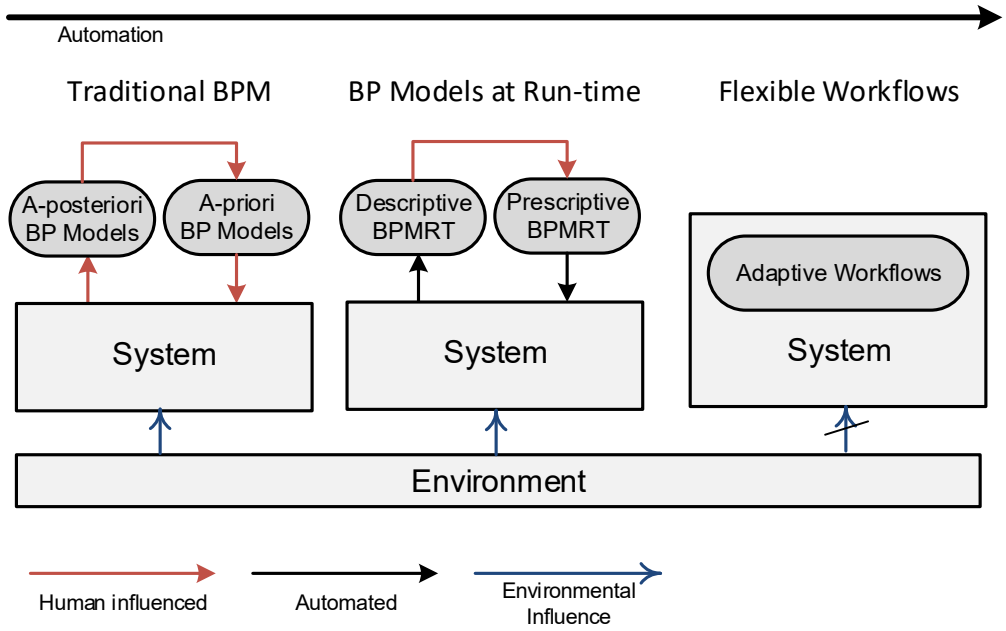


Fig. 3.3 Conceptual Differences of the Approaches

In order to sufficiently address the need for adapting to changing demands and for shorter BPM life cycles this thesis proposes a balanced compromise between these two sides: In complex and high-level domains like BPM a separation of concerns (system monitoring vs. adaptive reasoning vs. system modification) is preferred which can be achieved through a differentiation between descriptive MRTs and prescriptive MRTs. This approach is conceptually shown in the middle of Figure 3.3 in relation to the other two approaches. While it allows for more automation than the traditional approach through causally connected models it still allows for outside influences and human interaction during adaptive reasoning. In the following the focus of this thesis is on descriptive MRTs for the BP domain (DBPMRTs), i.e. this thesis focusses on merging the BPM lifecycle phase of diagnosis into that of enactment. That means in particular: (1) the specification of DBPMRTs, (2) the associated concern of system monitoring to establish and maintain the causal link from BPMS to DBPMRT at run-time, and (3) direct and automated reasoning based on the DBPMRT in the form of performance prediction. For each of these areas individual limitations of relevant state of the art work is discussed.

3.2 Descriptive Business Process Models at Run-time

The role of Descriptive Business Process Models at Run-time in the concept motivated in the previous section is that of a BPMS abstraction to enable BP domain reasoning. Depending on the reasoning type and approach the form and extent of a descriptive model can vary greatly, e.g. historical performance information is required for sensitivity analysis or optimizations [72, 131], or current state information is required for system adaptations [191]. In order to be used for most BP reasoning techniques, a generic descrip-

tive reflection of the system will have to capture the main aspects, *past* and *current*, of a BPMS, namely (see Section 2.1 on page 16): (1) the *control-flow* of the observed BPs, (2) the *organisational* perspective (which includes shared resources and roles across multiple BPs), and the (3) *performance* perspective[2]. A model language supporting the expression of these aspects will be also occasionally described as *"holistic"* in the remainder of this thesis. In this gap analysis section the "holistic" modelling capabilities of a selection of prominent state-of-the-art and state-of-the-industry modelling languages from three different domains are discussed: (1) Prominent BP languages used in BPM industry, (2) Worklfow domain languages, and (3) Abstractions or languages used in the domain of Process Discovery.

Business Process Management Languages

BP languages used in the BPM domain such as BPMN [168] or EPC [207] are used predominantly as prescriptive language with the purpose of documentation or communication (see Section 2.2.1). Since they are serving a different purpose they cannot fully reflect the run-time system in a descriptive way. In general such a-priori standards lack in: (1) providing modelling capabilities for holistic BP reflection, i.e. they are prescriptive model standards with which often only control-flow and roles can be modelled but not resources and performance, and/or (2) addressing specific run-time concerns, e.g. reflecting the current state or dealing with change in the environment. There have also been initiatives such as Business Process Runtime Interface (BPRI) [166] and Business Process Query Language (BPQL) [164] to introduce diagnosis standards into BPM. However, they lack maturity or are discontinued and have so far failed to produce standards adopted by industry. Instead of a coherent and holistic descriptive approach, in BPM practise there is a clear distinction between run-time analysis and offline diagnosis (see Figure 2.7 on page 24).

Run-time analyses in BPM are often referred to as Business Activity Monitoring (BAM) approaches, e.g. [50, 68, 130, 149, 282]. However, they are highly customisable and require modelling input (e.g. definition of KPIs/PPIs, BP Model) and thus cannot handle a dynamically changing context. Additionally, they are mostly focussed solely on the performance perspective or other singular aspects of the process, e.g. KPI/PPI violation (see 2.5.1), path prediction, or violation monitoring, and as a consequence use languages specific to these singular aspects. One notable approach that uses a prominent BP domain language in a BAM (i.e. run-time) context is presented by Friedenstab et al. [68]: A BPMN extension is proposed which allows for the performance perspective to be reflected in the BP model.

The other type *Process Intelligence* describes diagnosis or analysis types in the traditional a-posteriori way. Being on the BP abstraction they are mostly utilizing BP domain languages. The observed system state is taken into account for the diagnosis either via

[2]The case/data and functional perspectives are not considered in the scope of this thesis since they are extremely dependent on the use case and implementation of the BP and therefore not easily generalised.

modelled expert knowledge, e.g. [72, 178, 180, 189], or via structured models in the form of offline event logs [131, 195, 259]. Additionally, these approaches are often combined with user modelled information, e.g. organisational structure, problem space, and are generally carried out on a strategical level with the purpose of *process reengineering*, e.g. for what-if analysis [73], sensitivity analysis [69], bottleneck detection [195], business impact analysis [189]. If the analysis is carried out in an independent fashion without modelled input this falls into the category of Process Discovery which is discussed in a separate later part of this section.

Workflow Languages

Workflow languages such as ADEPT$_{flex}$ [191], A$_{GENT}$W$_{ORK}$ [154], and YAWL (Yet Another Workflow Language) [245] were created with the purpose of making BP management more automated, i.e. bringing problem space and solution space together. However, because of legacy systems as well as conceptual and fundamental reasons discussed in Section 3.1.2 they have only been adopted successfully in easily automatable domains. With regards to their capability to represent descriptive business models at run-time, workflow models have the benefit of being designed for run-time purposes. As such they are able to deal with all or most run-time reflections and can in the case of adaptive workflow models (e.g. ADEPT$_{flex}$, A$_{GENT}$W$_{ORK}$) even deal with various extents of run-time flexibility on the type level (see Section 2.3). Two of the downsides with regards to the topic is that adaptive worklfow approaches are (1) on a different abstraction level and (2) assume complete governance of the processes execution. The latter downside is negligible from a language perspective since it has only ramifications for the causal connection between system and model (see Section 3.3). In contrast, the abstraction gap leads to lack of representation capabilities with regards to other perspectives than the control-flow perspective. This stems from the focus of adaptive workflow approaches on the enactment of change and all entailing difficulties. As a result other perspectives such as organisational or performance are regarded as secondary, i.e. passive.

Descriptive Abstractions or Languages used in Process Discovery

Since business processes are enacted in a way that allows for unforeseen exceptions, deviations, and evolution steps Process Discovery became a very prominent research field in BPM diagnosis phase that (from a models@run.time perspective) helps to establish a causal connection from system to model. The languages used in traditional process discovery techniques are predominantly General Purpose Languages (GPLs) like Heuristic Nets [278] or Petri Nets [172] (see Figures 2.17, page 42 and 2.15, page 39). Their benefit is that these formal languages of BPM theory make an evaluation more comparable. A disadvantage is that they are on a different abstraction level an do not posses the basic BP domain-specific elements such as *Activity*, *XOR/AND-Branch*, or *Role*. While tools like the ProM framework [253] allow for a translation of Petri Nets or Heuristic Nets to and from

BP-domain languages like BPMN and YAWL. Although the success of this is not guaranteed since the abstraction differences are great and the translation is a non-trivial matter concerned with execution-equivalence. To bridge the gap, somewhat block-structured Petri Nets [26, 126] can be used as an abstraction that simplifies the problem of transformation. While this might serve as mitigation strategy languages used in Process Discovery do not serve as a good basis for a descriptive BP run-time model due to: (1) the extreme abstraction gap (including missing other BP perspectives, e.g. performance, organisational) and (2) missing representations for instance state (only Heuristics Net) or type evolution.

Summary

The findings of the gap analysis for descriptive business process models at run-time are summarised in Table 3.1. The content is explained further in the following paragraphs.

BP Languages: A selection of prominent state-of-the-art and state-of-the-industry modelling languages from three different domains were analysed with regards to their capabilities section the "holistic" modelling capabilities. The investigated languages are a representative selection of all existing graph-based modelling approaches and their extensions based on prominence and relevance. In particular, the representatives of BP Languages in Business Process Management are Event-driven Process Chain (EPC) [207] and *Business Process Model and Notation* (BPMN) [168] as well as the BPMN extension of Friedenstab et al. (BPMN+) [68] which allows for the definition and capturing of the performance perspective in the BP model. The most prominent abstractions used in the domain of Process Discovery are the general purpose languages *Petri Nets* (PN) [172] (and its extensions) as well as *Heuristic Nets* (HN) [278]. In the domain of worklfow the languages *Yet Another Workflow Language* (YAWL) [245] and ADEPT$_{flex}$ [191] are representative of adaptive workflow languages. With the exception of the languages of the Process Discovery domain (PN, HN) and the extended BPMN approach (BPMN+) these languages have been primarily developed for planning or execution purposes.

Abstraction Perspectives : The investigated languages have differing support for the abstraction perspectives of interest[3], namely: Control-Flow (Ctrl.-Fl.), Organisation/Resources (Org.), and Performance (Perf.) (see Section 2.1):

- Control-Flow: With the exception of HN and PN all other languages can capture control-flow via BP-domain specific elements, e.g subprocess, activity. HN and PN are general purpose abstractions of a BP's control-flow.
- Organisational: The organisation hierarchy including roles and resources or other representatives like organisational unit and position (see EPC) can be represented

[3] As stated earlier in this section, it is focused on main perspectives that are BP-domain specific and can be generalized in this context.

Table 3.1 Overview Gap Analysis Descriptive Business Process Models at Run-time; Legend: (-) not supported, (∘) somewhat supported, (+) mainly supported

BP Language	BP Type Perspectives			BP Instance State			BP History		
	Ctrl-Fl.	Org.	Perf.	Ctrl-Fl.	Org.	Perf.	Ctrl-Fl.	Org.	Perf.
EPC	+	+	+	∘	∘	−	∘	∘	−
BPMN	+	+	−	−	−	−	−	−	−
BPMN+	+	+	+	−	−	+	−	−	+
PN	∘	∘	∘	+	∘	−	−	−	−
HN	∘	∘	∘	−	−	−	−	−	−
YAWL	+	+	−	+	∘	−	−	−	−
ADEPT$_{flex}$	+	−	−	+	∘	−	+	−	−

by all analysed BPM languages (EPC, BPMN, BPMN+), the workflow language YAWL, and in an abstracted fashion also by the general purpose languages HN and PN (the resource-role relationship can be modelled via sub-petri nets). ADEPT$_{flex}$ does not support an organisational perspective since it only focusses on the control-flow.

- Performance: Descriptive temporal or probabilistic information in form of performance parameters (PPIs or KPIs) can be modelled with EPC, the BPMN+ extension, and partly with HN (only probabilistic PPIs, e.g. decision probability) or PN (e.g. CPN with timed transitions). YAWL, ADEPT$_{flex}$, and BPMN do not specifically support the modelling of KPIs, although the BPMN allows for extensions to do that (see BPMN+).

State of Perspectives: The current state of the respective perspectives can only be described with some of the languages, in particular:

- Control-Flow: It is common to capture the state or state changes of instances in an event log. PN (in an abstracted fashion) and the workflow languages YAWL as well as ADEPT$_{flex}$ have possible representations to represent the BP instance state within the model. The EPC model language has no particular state representation (only via event declarations) but together with EPC-based BPMS this state information is still accessible via event logs. While BPMN (and BPMN+) has no concrete specification of a control-flow state it can potentially capture the state for activity and data (but this needs to be meta-modelled by an extension). HN is not capable to represent a BP instance state.

- Organisational: The organisational state is the current association between activities and resources (current resource occupancy). PNs have the notion of state (via tokens in places) and can abstractly capture the resource perspective. As such the organisational state can theoretically (and non-BP-domain specific) be modelled via PN. The other languages have also no specific provision to model this, however with the possible representation of the BP control-flow state (see YAWL, ADEPT$_{flex}$,

EPC) this is indirectly achieved. HN, BPMN, BPMN+ have no possibility to model the resource state.

- Performance: Amongst the analysed languages the current state of the KPIs/PPIs can only be expressed by the BPMN+ approach (which was developed for this purpose). With regards to BPM and worklfow languages (EPC, BPMN, YAWL, ADEPT$_{flex}$) the capturing of modelled performance parameters is usually considered a purpose of the BPMS or BAM solutions and as such not part of language specifications.

History of Perspectives: The history of the instance state is captured by the transactional events during the execution. On the type level, change events are only available in specialised holistic frameworks that govern BP changes, e.g. with ADEPT$_{flex}$ all type changes af the control-flow are captured with the relevant change characteristics. The other languages do not inherently support modelling capabilities to capture the history of BP types, however, this is often (and especially in prescriptive languages) imitated by manual (e.g. through different files) or automatic versioning. One such automatic example is the EPC framework where a design history captures the different design versions of the organisational and control-flow perspectives [218]. To capture the history in a non-model-based way is not a desirable approach since references between different perspectives become either broken or need to be completely cloned. BAM solutions like BPMN+ support the history of the performance perspective but do not allow for capturing the history of the other type level perspectives. In conclusion, model-based support to capture the evolution or history of the type levels independent of the source and extent of the change is not well supported by current BP-domain languages and thus constitutes a significant gap in current state of the art and industry.

3.3 Process Discovery at Run-time

Through process discovery a causal connection from the system to the discovered BP Model can be established. Related relevant work in process discovery has been discussed in Section 2.4 and is the basis for this gap analysis.

3.3.1 Gap Analysis: Process Discovery Algorithms

The general problem is that due to uncontrolled deviations from the planned BP during implementation and enactment the real BP needs to be discovered directly from execution event logs (see Section 2.2.2). The problem of process discovery and its challenges (see Section 2.4.1) are well researched with a very large number of publications in this area. The most relevant techniques (see Section 2.4.2) have, however, individual shortcomings with regards to functional requirements necessary to establish a causal connection from BPMS to discovered BP model:

1. **No Infrequent and Incomplete Logs**: A reality in real-life scenarios are infrequent (noisy) and incomplete logs. To establish a strong causality an algorithm must be able to interpret any given real-life scenario. However, many algorithms cannot deal with incomplete and noisy logs, e.g. abstraction-based and other factual algorithms not employing a heuristics-based approach [45, 249], or region-theory based algorithms [132, 243].

2. **Representation not on BP Abstraction Level**: There is a noticeable difference between business process specifications at design-time and the representations of business processes discovered from logs. Whereas prominent standards for business process models, e.g. BPMN and EPC, are BP-domain-specific, the results of process discovery algorithms conform to general purpose representations like Petri Nets, e.g. [243, 249], or other abstract languages such as Causal Net, e.g. [278], or fuzzy models, e.g. [89]. For a business analyst, the comprehension of these general purpose languages for decision making is a difficult task, because (1) these are of a different representation and abstraction level than what she is familiar with and (2) the mapping between the process modelled at design-time and the discovered process model at run-time can be difficult to establish since they both conform to different languages. This can be mitigated by language transformations which are however not guaranteed to produce a sound BP-domain model since they reside on a different level of abstraction. Also, techniques in the area of process discovery almost exclusively focus on the discovery of the control-flow perspective and neglect other perspectives such as resources and performance (see Section 2.1). Techniques that mine information about these perspectives are mostly associated with the *model enhancement* discipline (introduced in Section 2.2.3) and discussed in more detail in Section 2.5.3. Furthermore, representations such as Petri Net, Causal Net, etc. can not be easily used for further analysis and decision support since they have difficulties to express perspectives other than of the control-flow, e.g. resources, performance. This limitation of expression is fittingly summarised by Aalst in [230] who concludes *"The world is not a Petri Net"* and motivates the need for a more representative, BP-domain-specific solution.

3. **Non-Deterministic or Non-Automatic**: Generally, non-deterministic, e.g. genetic algorithms [26, 147] or neural nets [39], as well as techniques that require manual effort, e.g. [243], are not suited for the discovery of causally connected run-time BP models. Although non-deterministic types of analyses like the genetic algorithms do not subscribe to any of the above mentioned shortcomings, the simple fact that no correct or stable output can be guaranteed for the same input, makes them unsuitable to maintain a causal connection in a real-time setting; The same applies for approaches requiring manual effort.

Other limitations of process discovery algorithms exist but are rather dependent on other aspects of the use case than the establishment of a causal connection at run-time. One

such example is the balance between over-fitting and under-fitting: Since often the quality of the solution is dependent on what is more important for a specified use-case, e.g. under-fitting might be preferred to discover human readable non-spaghetti nets - in other occasions the discovery of extremely accurate over-fitting models might be desired. Another such use-case dependent limitation is that of footprint abstraction: Most of the process discovery approaches are mainly based on footprints representing direct activity successions, i.e. some form of a local *directly-follows* relation, e.g. [126, 249, 278]. Global relations, e.g. activity x_1 is eventually followed by activity x_2, are either not taken into account or are only used to identify and reflect special behaviour, e.g. non-free choice constructs [279]. Especially with regards to the discovery of parallel constructs the eventually follows relation has a clear benefit in comparison to the directly follows relation: all "follows" combinations of the elements of two parallel paths are easier to observe when they are required to be "eventually" rather than "directly". This will be discussed in more detail in Section 5.2. The author argues that results can be improved and computation costs reduced by basing the discovery on global relations (see Section 5.8.1).

In the context of this gap analysis the Inductive Miner (IM) [126] together with its extensions [127, 128] to deal with incomplete and noisy logs deserves special mentioning: It is the only algorithm - to the best of the authors knowledge - that does not subscribe to any of the three main limitations that apply when PD algorithms are used for establishing a causal connection during run-time. However, still two limitations remain: (1) The IM is based on local rather than global relationships and (2) it has a potentially exponential run-time when considering incomplete logs because the involved constraint solving is an NP-hard problem for which an SMT[4] solver is used [128].

3.3.2 Gap Analysis: Online Process Discovery

Business processes can stretch out for months, have a very high instantiation frequency, or both. For a slow and long executing BP the traditional process discovery approach might temporarily create causality from system to the discovered model; this is not the case for BPs with very frequent state changes. Here, the causality from system to model is already "outdated" when the discovery analysis has finished. Another additional challenge in the BPM domain which is not address by traditional process discovery algorithms is that of uncontrolled deviations as elaborated earlier in Section 3.1. Those can manifest themselves in many different forms, e.g. (1) a predicted condition does in reality never apply, (2) exceptional or unforeseen conditions occur that demand ad-hoc adjustments in the execution flow, or (3) a gradual or abrupt deviation from planned BPs towards a more applicable/optimal (in real life) execution flow takes place. The first two deviation examples are addressed by the more sophisticated process discovery algorithms of recent years which can deal with noisy and incomplete logs (see discussion above). However, during recent years and due to the advancement of Big Data, a frequently changing en-

[4] Satisfiability Modulo Theories

vironment, and fast machine guided processes, the latter deviation example of continuous and uncontrolled BP evolution has gained more relevance. These observations build the foundation of (initial) research in the fields of concept drift in the business process domain for which recent research is reviewed in Section 2.4.3). In reality only a "...few processes are in steady-state and due to changing circumstances processes evolve" [138]. This view stands in contrast to the view of traditional PD approaches where BPs are assumed to be steady-state. In Section 2.4.3 different solutions to address concept drift are discussed, e.g. [21, 23, 34, 274–276], including their individual limitations. The biggest of which with regards to descriptive BP run-time models is that their purpose is essentially different: they focus on detection and localisation of concept drift and are rather pre-processing methods to identify where to split logs so that they can be individually analysed by a traditional PD algorithm. The sole detection and localisation of these drifts on static event logs are not sufficient for maintaining a causal connection from system to model at run-time.

Approaches of Online Process Discovery as discussed in Section 2.4.4 go a step further. They work on an online stream of events rather than on an offline event log. The conceptual idea is that of carrying out Process Discovery with Complex Event Processing techniques (see Section 2.2.3). That means in the context of models at run-time, immediate processing of state changes (events) when they occur to information of an higher abstraction level (BP models). The motivation is to have a run-time reflection of the employed processes based on up-to-date rather than historical information. As discussed in the respective Section 2.4.4 online process discovery algorithms have to deal with two additional challenges as opposed to the traditional process discovery algorithms: (1) anticipation of concept drift, i.e. reflecting new behaviour as well as forgetting old behaviour, and (2) potentially processing of an infinite amount of events at high frequency, i.e. scalable with regards to the amount of events occurring. In the state of the art Section 2.4.4 it has been shown that incremental discovery approaches [37, 119–121, 219] address the challenges of online process discovery, albeit only partly:

- Incremental process mining [37]: Newly observed behaviour is added, but once observed behaviour will stay valid indefinitely despite it becoming outdated, i.e. no support for real concept drift since revolutionary changes (see Section 2.3) are not supported.

- Incremental workflow mining [119–121, 219]: Additonal to the same short-coming of incremental process mining, it also requires human interaction, i.e. not applicable in real-time environment.

A contrast to the incremental approaches are that of Streaming Process Discovery as discussed in the work of Burattin et al. in [27]. It discusses different techniques for using event streams for process discovery (e.g. sliding window, FIFO queue, ageing, lossy counting) and generally provides a solution that addresses the challenges of online process discovery. However, since it is based on the HeuristicsMiner its shortcomings apply

for this online approach as well, namely the BP-independent representation, the sole focus on the control-flow, and the reliance on only local relations between activities. To the best of the authors knowledge the only other streaming process discovery approach is that of Maggi et al. [138], but since it is discovering declarative process models (using the sliding window and lossy counting) it is outside of the scope of this thesis.

Summary

The findings of this section are summarised in Table 3.2. The used abbreviations as well as the content are explained further in the following paragraphs.

Approaches: Different types of approaches that are compared with regards to the challenges of BP model discovery at run-time: (1) Traditional Process Discovery (TPD) represent the static offline process discovery approaches based on Logs (see previous Section), (2) Traditional Process Discovery with Concept Drift Detection (TPD+CDD) is the combination of both fields, i.e. using a concept drift algorithm to detect where to split the log into sub-logs which are then individually analysed by a traditional Process Discovery algorithm, (3) Incremental Process Discovery as proposed in [37], (4) the approach of Maggi et al. (MEA) [138], (4) Incremental Workflow Discovery as proposed in [119–121, 219], and (6) the approach of Burratin et al. [27].

BP Representation: The BP representation of an approach decides what information can be retrieved and what further analysis can be carried out. The domain of process discovery is solely focussing on the control-flow perspective which as discussed in Section 3.2 is not sufficient. Furthermore, is the type of BP representation important: While traditional approaches (TPD) have also advanced towards discovering BP domain models, that is not the case for online process discovery approaches: Either they are based on declarative models (IPD and MEA) or on non-domain-specific models (IWD and BEA) such as Petri-Net and Heuristic Net.

Supported Input: While the traditional discovery approaches (TPD, TPD+CDD) work on complete logs, the online approaches (IPD, MEA, IWD, BEA) can operate on a stream of events. However, only the BEA can handle noisy or in complete logs, i.e. she is the only one able to handle real-world BP scenarios.

Concept Drift: Supporting concept drift is an important requirement to maintain the causal connection from system to BP model. While traditional approaches (TSP) assume to be observing an unchangeable process, i.e. do not support concept drift (−) the incremental approaches (IPD and IWD) support concept drift to some level (∘) and the other approaches (TPD+CDD, MEA, BEA) even fully (+).

Table 3.2 Overview Gap Analysis Process Discovery at Run-time

Approach	BP Representation		Supported Input		Concept Drift	Level of Automation
	Type	View	Type	Inc./Noise		
TPD	diverse	Control-flow Perspective only	Log	varies	–	varies
TPD+CDD	diverse		Log	varies	+	∘
IPD	Declarative		Stream	–	∘	∘
MEA	Declarative		Stream	–	+	+
IWD	Petri-Net		Stream	–	∘	∘
BEA	Heuristic Net		Stream	+	+	+

Level of Automation: Similarly, the level of automation is important to maintain the causal connection in real-time systems: while all approaches are at least semi-automatic (∘), i.e. only need some manual steps, others are able to operate fully automatic (+), i.e. without any manual steps.

3.4 Process Performance Prediction at Run-time

Traditional reasoning in BPM is mostly based on the analysis of state transition events, especially in the discipline of decision support (see Sections 2.2.3, 2.2.3, and 2.5). Part of an organisation and its business processes are its goals which are usually monitored through quantifiable measures expressing the performance of organisational units and processes, i.e. Key Performance Indicators (KPIs). Process-centric measures are also called Process Performance Indicators (PPIs), e.g. number of ordered items, rate of cancel requests. Evaluating KPIs and PPIs against their target values support the process of decision making in the BP lifecycle and they are often basis to higher level analysis such as bottleneck detection [195], or SLA violations [285], or performance prediction.

Current solutions for monitoring PPIs require modelling effort to specify their aggregation, and computation methods (based on the event data), e.g. [68, 130, 149, 282]. This approach is not applicable for dynamically changing processes. Here, the PPIs have to be of a generic nature and automatically adapt to the underlying BP, e.g. resource utilisation, end-to-end processing time.

With regards to the prediction of these PPIs there is two different general approaches: (1) a quantitative/statistical analysis of timeseries as is done traditional BI solutions [24, 29, 178, 205, 206], and (2) that of simulation, where a model of a system is used to "enact" the same or similar behaviour. Since the focus of this thesis is on run-time BP models and proving their worth for facilitating pro-active decision support and many other higher-level analyses, e.g. what-if, sensitivity analysis, and optimisation, the latter approach is of particular interest: It is considered to be versatile, imposes only a few constraints, and produces results that can be interpreted in a similar way than the ones of the real system [232]. In the BP domain it is particularly common to utilise the discrete-event simulation (see Figure 2.21 on page 57) due to its congeneric characteristics. Of this, again two

different types exist: (1) steady-state simulations and (2) short-term predictions.

Steady state simulations are based on the assumption that the system converges to the same balanced state if you let it run long enough (unless the resources cannot cope with the demand). Examples of this approach are [72, 131, 180, 189, 290]. While steady state simulations in their different implementations are especially of importance for strategic analysis like what-if, or sensitivity analysis, they neglect the current fine-granular state of the system, i.e. they start with an "empty" system.

On the other hand, short-term (or transient) simulations start to simulate from exactly the system's fine-granular state, and can more reliably answer questions about the immediate future of the BPs. This information can be used for more accurate higher-level decision support support. One such example is *adaptive reasoning* [62] as conceptually discussed in certain EDBPM publications, e.g. [268–270]. Another example is automated optimisation as prototypically shown in [221]: Here the future performance of every business process variant, which was computed via using simulation, represents the respective fitness of the individuals for the optimisation. However, all of these approaches have so far only been researched conceptually or in a prototypical manner without evaluation on real-life use cases or being applied in industry.

In the context of BP models at run-time, a more suitable and evaluable reasoning use-case of transient simulation is the short-term prediction of a BP's performance. The more accurate and detailed the BPMS is reflected as descriptive BP run-time model, the better prediction result can be expected. Although this technique potentially yields improved results (see Figure 2.22 on page 59) than statistical methods such as trend analysis or steady state simulations, only the approaches by Rozinat et al. in [198, 200] addresses the challenge of transient PPI predictions via simulation. The first approach in [200] is based on *modelled* BP design information and extracts the state in a non-generic way directly from the system (but not from the event log). The event log is only used to compute performance input and decision point information. In [198] it is shown that the design information (control-flow and resource perspective) can also be extracted from the log. However, the current state is neglected and due to the process discovery approach used (α-algorithm) noise and incomplete behaviour in the log may lead to the discovery of invalid and non-simulatable models (see Figure 2.15 on page 39). Furthermore, neither this nor any of the other simulation model discovery approaches is able to deal with dynamically changing processes, monitor more than one process, or is applicable "online" on an event stream to allow for (near-)real-time results.

Summary

Table 3.3 summarises the gap analysis findings of this section with regards to performance prediction at run-time. In the following paragraphs used abbreviations as well as the content are further explained.

Prediction Approach: We distinguish between five different types of approaches: (1) Analytical Timeseries Analysis (ANA-TA) comprise traditional BI methods based on correlations, extrapolations, e.g. [24, 29, 205, 206], (2) Analytical mathematical models (ANA-MM) which comprise methods that abstractly emulates structural behaviour with sophisticated mathematical models such as FMC-QE [178], (3) Steady-state Discrete Event Simulation (DES-SS) [72, 131, 180, 189, 290], (4) Transient Discrete Event Simulation (DES-T) by Rozinat et al. [200], and (5) Transient Discrete Event Simulation based on Event Logs (DES-TE), an extension of approach (4) and partly based on event information [198].

Source of Levels: Whether information on the two granularity levels is used and if so from where it is extracted:

- Instance Level: Neither the traditional BI approach ANA-TA, nor the steady state approaches ANA-MM and DES-SS use fine granular instance state information (i.e. not applicable "N/A"). The two transient approaches (DES-T and DES-TE) use them but do not extract this information from the events but rather directly from the "BPMS" through a generic but not further specified interface.
- Type Level: While ANA-TE is not based on structural information at all ("N/A"), approaches ANA-MM and DES-T require this information to be modelled ("M"). The DES-TE solution allows to discover the simulation model from an event log ("D"). The DES-SS is a special case - here it depends on the individual solution: Some solutions are originally based on modelled simulation models, e.g. [72, 189], but some solutions (e.g. when based on YAWL like in [290]) also may allow the simulation model to be discovered from an event log, i.e. "M/D".

Resource Representation: The structural view on resources varies for the different approaches: For solutions of the ANA-TA type it is not explicitly included (−) since they cannot replicate queueing behaviour; Others like DES-T and some of the DES-SS solutions (e.g. [72, 189]) do not support shared resources across multiple BPs but instead model resource availability or activity waiting time through probabilistic values (∘); The others allow for a simulation model with shared resources across multiple BPs (+).

Special Characteristics: Reasoning on real-life stream data requires the support of the following characteristics:

- Incomplete/Noisy: Simulations based on the descriptive BPMS state utilise discovery algorithms (see Sections 2.4 and 3.3) which should be able to deal with incomplete behaviour and noise in the event logs (see Section 2.4.1). For ANA-TA, ANA-MM, and DES-T this is not applicable (N/A). DES-TE is based on the α-algorithm and thus does not support incomplete or noisy behaviour (−). For DES-SS it depends on the specific approach and is supported in some cases (−/+), although it has not been explicitly mentioned in the publications.

Table 3.3 Overview Gap Analysis Process Performance Prediction at Run-time

Prediction Approach	Source of Level		Resource	Online Characteristics		
	Instance L.	Type L.	Represent.	Inc./Noise	Conc. Drift	Stream
ANA-TA	N/A	N/A	−	N/A	∘	+
ANA-MM	N/A	M	+	N/A	−	−
DES-SS	N/A	M/D	∘/+	−/+	∘	−
DES-T	BPMS	M	∘	N/A	−	−
DES-TE	BPMS	D	+	−	−	−

- Concept Drift: Run-time performance predictions must be able to handle process change, i.e. concept drift. This is indirectly supported by ANA-TA on a non-structural level (∘) but by none of the others (−).
- Event Stream: Similarly run-time performance predictions should be causally connected to the system, i.e. work on an event stream. While most solutions of ANA-TA are capable of doing so (+) this is not the case for the other approaches (−).

3.5 Summary

Based on the state-of-the-art literature review from Chapter 2, this chapter discussed the fusion of the BPM and MRT domains as well as the resulting gaps of current research in the context of Descriptive Business Process Models at Run-time.

First, in Section 3.1 the feasibility of employing MRT concepts in the domain of business processes was discussed in general and two special characteristics of the BP domain were identified that are uncommon in conventional MRT concepts:

1. the unusually *high abstraction gap* between BP model and system which are manifested in all phases of the BP lifecycle, and
2. *uncontrolled deviation* which occur not only in the implementation and configuration phase but also in the enactment phase of a BP's lifecycle.

Then in Section 3.1.2, it was discussed what implications that has for potential solutions for Business Process Models at Run-time and why holistic solutions like adaptive workflows do not meet those challenges due to one major simplification that usually does not apply in the BP domain: *The assumption of closed-loop, self contained systems without external changes sources.* Finally, a two phased concept that consists of two run-time models, descriptive and prescriptive, was proposed in order to account for the previously identified special characteristics of the BP domain: high abstraction gap and uncontrolled deviations.

This concept is the base assumption for the gap analyses for the individual goals of this thesis. The following individual gaps of current state of the art were identified:

1. Descriptive Business Process Models at Run-time (Section 3.2): It was shown that prominent languages from the three relevant domains of BPM, workflows, and process discover do respectively support only a subset of all essential descriptive or run-time information, i.e. type level information as well as their history and current state for the main abstractions of a BP: control-flow, organisation, and performance. The analysis showed that especially when considering the modelling capabilities to capture the history of type level changes (rather then instance level changes which are captured in a traditional event log) a clear gap exists for state of the art or industry languages.

2. Process Discovery at Run-time (Section 3.3): Here it was shown that no solution can address all requirements for a causal connection from system to model at run-time, i.e. bridging the abstraction gap, incorporating all perspectives (control-flow, resources, instance, performance), working on event streams with noisy and incomplete behaviour, dealing with changing processes (concept drift), and requiring no human interaction.

3. Process Performance Prediction at Run-time (Section 3.4): Here it was shown that no solution can predict PPIs based on abstracted BP type level and instance level information from an event stream with noisy, incomplete, and changing behaviour.

The gaps will be addressed in the subsequent contribution chapters.

Chapter 4

Descriptive Business Process Models at Run-time

The motivation of this chapter is to fulfil the first objective of this thesis, the *specification of a DBPMRT reference modelling language*, by addressing the gaps identified in the last chapter in Section 3.2. As a result, the proposed "holistic" model language is required enable the capturing of type level information as well as their history and current state for all main abstractions of a BP: control-flow, organisation, and performance.

Figure 4.1 shows the outline of this chapter and the relation to the remaining contribution chapters. In the first Section 4.1, the objective of specifying DBPMRTs is translated into concrete requirements that should be met by a modelling language for DBPMRT while considering the input of the gap associated analysis in Section 3.2. These requirements are essential characteristics a DBPMRT needs to comply to and represent the basis for the following sections. Then in Section 4.2 reference meta-models for a DBPMRT are proposed with the focus on capturing the holistic status of an observed BPMS. In Section 4.3, these reference meta-models are further enhanced to enable the capturing of the temporal evolution of the BPMS's status on the type level. This includes a discussion on the different meta-model compositions that are possible. In Section 4.4 the proposed reference meta-models for DBPMRTs are evaluated in a qualitative fashion against the previously established requirements. The proposed DBPMRT language's value of use is evaluated through its application in the subsequent chapters.

4.1 Identification of Characteristics for DBPMRTs

A language for run-time BP models has to support specific characteristics in order to meet the requirements of a run-time model. In Section 3.2 it was identified that current languages either cannot capture other perspectives than the control-flow, or lack in capturing the current state of an instance or the type level history or a combination of these. Based on these results a discussion on what is essentially required of a language in terms of reflecting different abstraction levels of states and changes in order to constitute a lan-

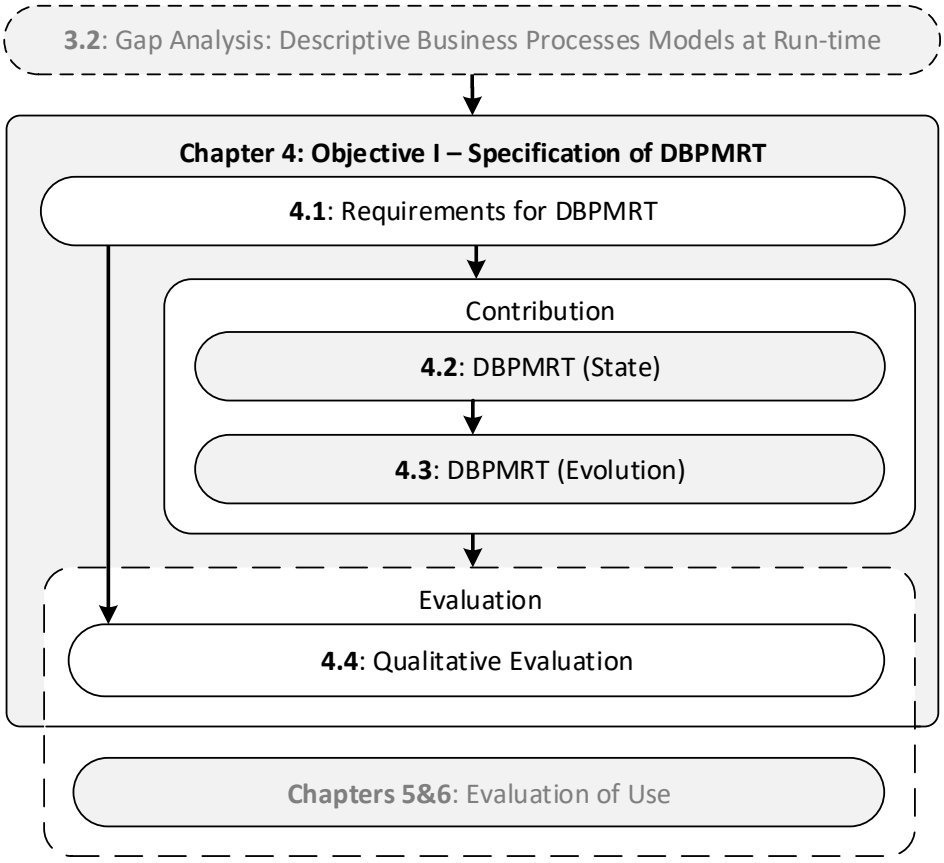


Fig. 4.1 Outline of Chapter 4

guage for DBPMRTs.

4.1.1 Characteristics for Run-time BP Models

The classifications of elements for general characteristics of run-time models, reviewed in Section 2.7.4 (Generalisation of Models at Run-time), is a good entry point to identify requirements for such a language. As pointed out in that section, the ratio of prescriptive and descriptive parts are dependent on the purpose of the model, e.g. monitoring MRT vs. executable MRT. But not only the ratio of these parts can be different also the run-time aspect, prescriptive or descriptive, can vary for the same model element types depending on the purpose. That is, an element type, e.g. an activity, can be of a prescriptive nature in one run-time BP model, e.g. execution standards like BPEL, but of a descriptive nature in another run-time BP model, e.g. a BP model extracted via process discovery (see Section 2.4). In all cases (descriptive, prescriptive, or both) the objective is to allow for or record changes on different abstraction levels (e.g. type level vs. instance level) and across different BP perspectives (e.g. control-flow vs. resources). The current state-of-the-art work classifying these *dimensions of change* in the BP domain is discussed in Section 2.3. Particularly of relevance with regards to the dimensions of change in the BP domain are the two approaches by Sadiq et al. [202] and by Schonenberg et al. [211], both defining different types of changes from a realisation perspective rather than a context

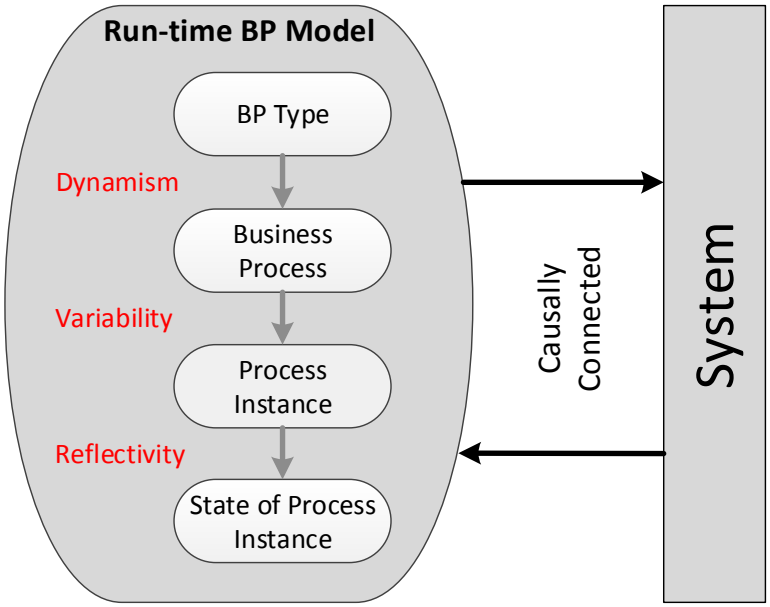


Fig. 4.2 Abstraction Levels of BP Changes

perspective. A summary is depicted in Figure 2.11 on page 31 which merges both classifications with regards to their abstraction (type vs. instance) and anticipation (build-time vs. run-time).

Since the focus of this thesis is on techniques that allow changes during run-time, only the associated abstraction level of the change is of importance, i.e. the granularity of a change. There are two abstraction levels of change that can be identified in both classification: (1) The change of the execution path of a process instance (instance level), in the remainder called *Variability*, and (2) the change of a complete business process definition (type level), in the remainder called *Dynamism*. Due to the focus of both approaches on process change, one abstraction level of change has not been regarded, yet: the fine-granular state change in a process instance, in the remainder called *Reflectivity*. We argue that a language for run-time BP models needs to be able to support these three dimensions of change (see Figure 4.2) in order to support any business process related purpose from business process monitoring to dynamic process optimisation.

A first conclusion to be drawn after identifying these three different dimensions of change is that reflective business process models, e.g. workflow models like ADEPT$_{flex}$ [191] that are executable and monitor the state of the system, are technically already adaptive run-time models. The inherent flexibility of BP modelling languages (e.g. via conditional or parallel behaviour) describes execution rules for the individual process instances, i.e. these BP models are adaptive run-time models for *process instances* but not for the BP type level. That means in the context of the differentiation between descriptive and prescriptive model parts by Lehmann et al. [129] (see Figure 2.31 on page 79), that the prescriptive part specifies the possible states and transitions of one process instance, the descriptive part describes the current state in the process instance, and the valid modifications of the prescriptive parts are the shifts of execution paths for the process instance dependent on

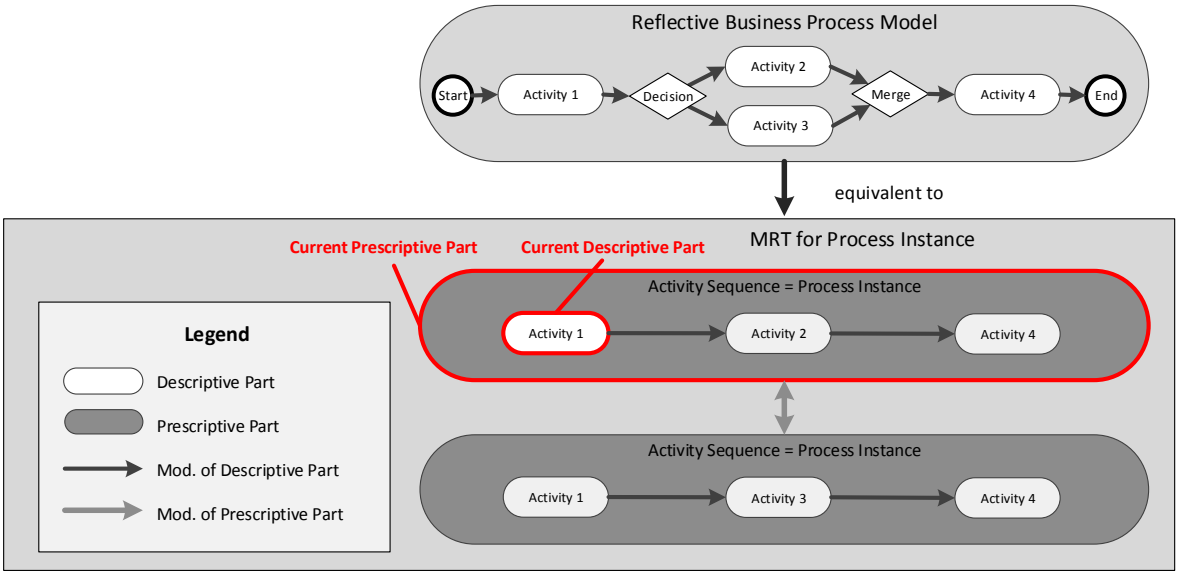


Fig. 4.3 Reflective Business Process Model as Process Instance Model at Run-time

the circumstances. This is shown in Figure 4.3. Fully automated workflow languages like ADEPT$_{flex}$ are a good example to show how important the abstraction level of change is, i.e. in terms of dynamic adaptation: On what level the MRT system captures change of the system and on what level the MRT system is governing modifications for the system.

With regards to general run-time BP models, the requirements are to be able to capture and govern change on the levels of *reflectivity*, *variability*, and *dynamism*. In addition to the desired change dimensions, run-time BP models also have to support standard business process modelling capabilities which is why the requirement of *expressibility* is essential, as well. Figure 4.4 shows all four general requirements for run-time BP models in an conceptual overview. In accordance with the distinction between prescriptive and descriptive MRTs discussed in Section 2.7.5 a similarly distinguished view for the domain of dynamic BPs is highlighted: While the light grey arrows display *prescriptive* connections between the abstraction levels, the stroked black arrows highlight *descriptive* connections between the levels. The fundamental difference between these relations is the following: (1) the prescriptive connection represents that the higher abstraction level defines the *potential* "solution" space according to which the lower level is substantiated at run-time (i.e., a dynamic BP language or BP type defines via goals, rules, etc. according to which the business process is substantiated; the BP model describes rules for the BP instances; the BP instance defines the parameters of the states it can take), (2) the descriptive connection represents the actual as-is situation in a run-time system (i.e. the BP instance state describes the BP instances current state; the monitored BP instances describe the actual BP; the captured BPs describe the evolution of the BP on a type level). While the prescriptive connection represents a definition of the lower level space, the descriptive connection is the concrete monitored substantiation on that level. Note, that on the top level two different abstractions exist: the one on the left (BP Type/Dynamic BP) defines all possible variations and adaptations of the BP model while the one on the right (BP Evolu-

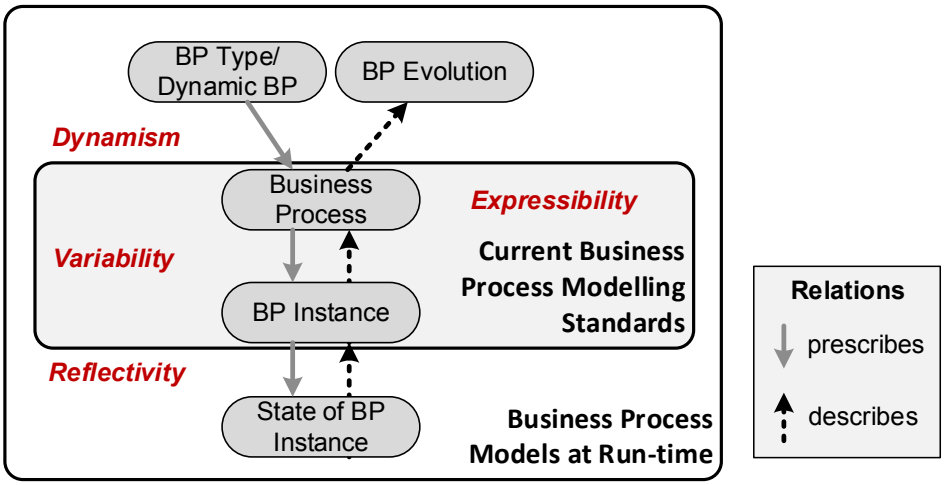


Fig. 4.4 Business Process Abstraction Levels and the Conceptual Location of the Four Required Characteristics for Run-time BP Models: Dynamism, Variability, Reflectivity, and Expressibility

tion) temporally describes the inhabited states and their development during execution. Also highlighted in Figure 4.4 is that normal BP standards such as BPMN [168] already inherently meet the requirements of variability and expressibility. Reflective workflow languages like ADEPT$_{flex}$ [191] additionally support the capturing of the lowest level, i.e. the BP instance state.

4.1.2 Concrete Requirements for DBPMRTs

In the context of this thesis the focus is on realising the descriptive connections, i.e. defining a language for DBPMRT which meets these four identified requirements. In the remainder of this section the concrete requirements for a DBPMRT are formulated.

Levels of Change

One general aspect of DBPMRTs is the support for representing change on the different abstraction levels. For development, deployment, and enactment as well as for the diagnosis, BP domain artefacts exist that operate on one or a number of different abstraction levels. Figure 4.5 shows these artefacts sorted by level of their abstraction. On the left all development and run-time artefacts and examples are displayed that are a product of design, implementation, and enactment of a BP: In particular, the BP type defined by goals and KPIs, e.g. through a specification document, is the basis for the designed BP model (problem space), e.g. with BPMN [168]; After the completion of the BP design it is then implemented by a BP execution language (solution space), e.g. BPEL [161], and deployed/compiled in order to be enacted by a BPMS such as SAP Netweaver BPM [287]; Internally, during enactment BP instances of the BP model are created, controlled/managed, and discarded. As a result BP events representing state transitions on the instance level are recorded and stored conforming to a specific event standard, e.g. XES [90]. During

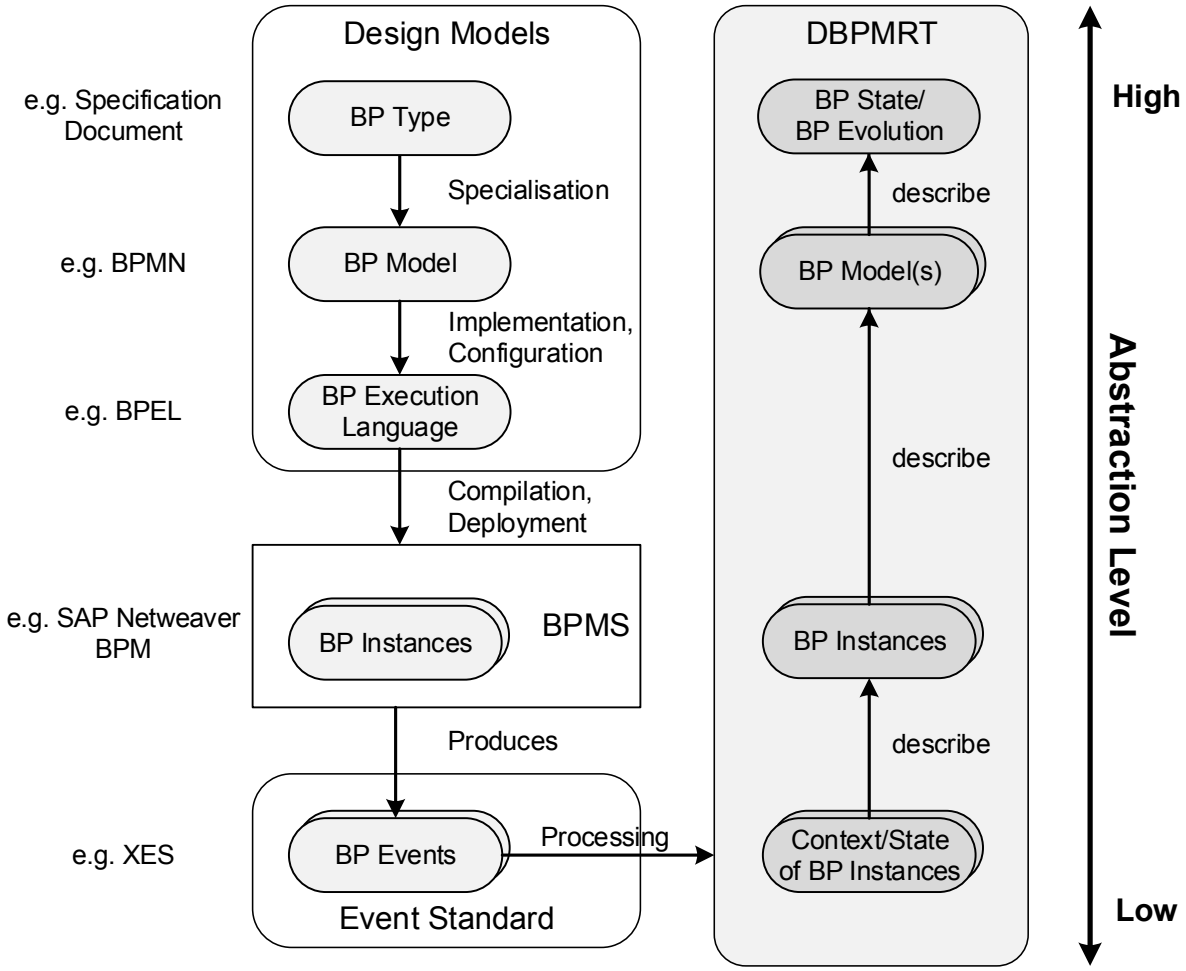


Fig. 4.5 BP-domain Artefacts Ordered by Abstraction Levels

that process the abstraction level continuously decreases.

In a similar fashion, a descriptive BP model at run-time (DBPMRT) is required to capture information and change on these abstraction levels (right hand side of Figure 4.5), i.e. the higher level data is inferred from/described by lower level data. In this context a language to model holistic DBPM@RTs has to support the following change level-related requirements[1]:

MR1 *Reflectivity:*
Reflectivity is the ability to represent fine-granular change in the system, i.e. a state transition (BP event) needs to be translated into, and reflected by, a valid update of the instance information in the DBPMRT. Examples are the current lifecycle state of an activity or which resource is executing this activity. In a broader context, information that is not directly reflecting the fine-granular state but represents aggregated information for a single instance can also be regarded as reflectivity, i.e. describing an instance's state. Examples are *end-to-end processing time* or *activity net working time* (see Section 2.5.1) of one single instance since they are each the result of at least two state transitions.

[1] MR = Model Requirement

MR2 *Variability:*
Variability is the ability to allow for flexibility on the instance level, i.e. the language supports the meta concept of individual instantiation. This means in practical terms for the DBPMRT that the information captured about observed instances can be generalised, represented, and accommodated by a single BP model. An example of this generalisation is, for instance, the inference of the actual control-flow from the respective activity sequence of each instance (this is the subject of process discovery techniques - see Section 2.4.1), or aggregated performance values such as the average activity net working time. With regards to the control-flow, prominent BP model languages support this requirement by default since they primarily design the control-flow and feature in that context *flexibility by design* [211] (see Section 2.3.1).

MR3 *Dynamism:*
Dynamism is the ability of business process model to represent change on the type level of the process. In the context of DBPMRT there are two different representations that can represent the change: (1) A *BP State* model that allows for changes on the type level but only captures the current as-is state and discards outdated information, (2) A *BP Evolution* model that records the evolution of the type information, i.e. an additional time dimension is included which allows to retain information about historic states of the BP rather then discard them. Examples of a BP state model is a BP simulation model capturing the complete context of the system to enable accurate predictions. A BP evolution model on the other hand is beneficial to analyse the development of a BP in order to find change patterns and cycles or simply to understand higher level changes of dynamic BPs.

Expressibility

In contrast to the levels of change the requirement of expressibility comprises concerns specific to the BP domain. Many different views exist about what information a BP model should be able to comprise which is usually dependent on the purpose of the language/ standard. In Section 2.1 it is discussed what different perspectives of a BP generally exist. The most prominent among them are the control-flow, resource, and performance perspectives. In order to allow for a generic and flexible solution which supports different purposes the principle of separation of concerns needs to be applied with regards to expressibility in DBPMRTs. That means, in particular, that these perspectives are required to be separated which essentially provides the following two benefits: (1) modular structure, i.e. complete perspectives can be added or removed; (2) simplified extension of the respective perspectives. Figure 4.6 shows these three BP perspectives, a set of representative elements, and the direct associations between them: (1) control-flow perspective on the left, (2) performance perspective (PPIs - see Section 2.5.1) in the centre, and (3) resource/organisation perspective on the right. Note that only selected elements on the

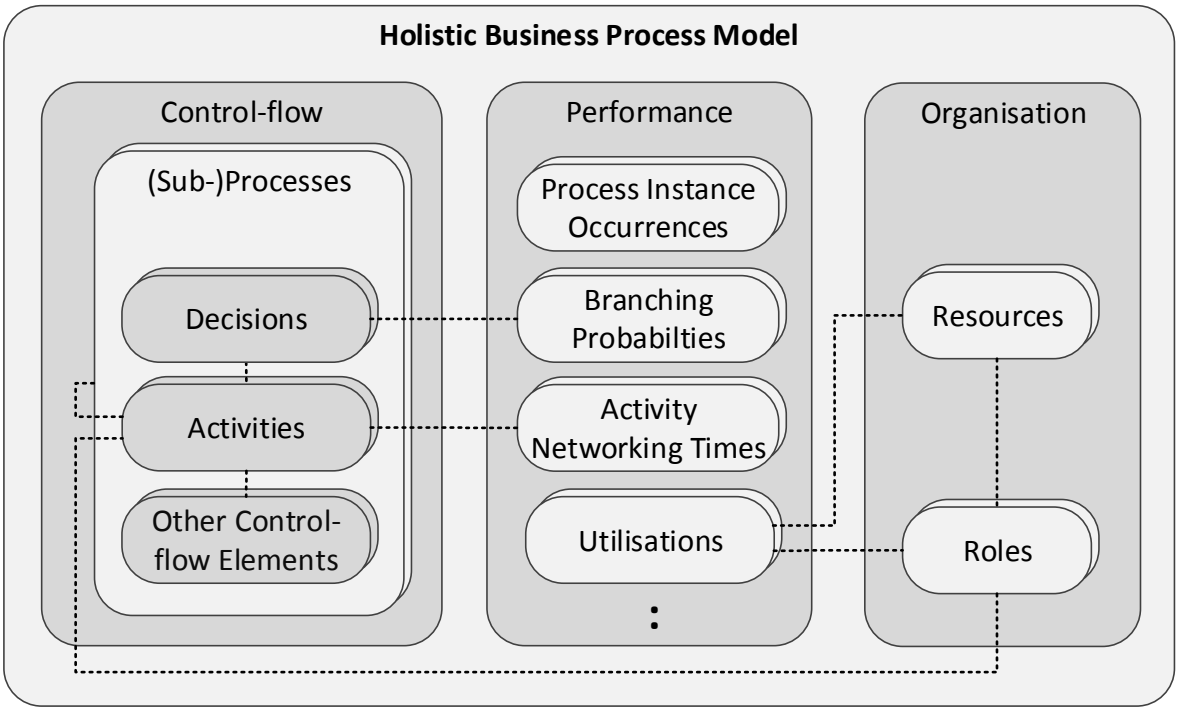


Fig. 4.6 Selected Element Types of a Business Process and Associations between them (Stroked Lines)

type level are included in the figure. If required, instance-specific data is also part of an holistic BP model. The expressibility requirements is summarised by the following description:

MR4 *Expressibility:*
Expressibility is the ability of a DBPMRT to express BP-specific requirements reflecting the purpose of the language. Influenced by the industrial use cases, the DBPMRT language should conform to the following concrete expressibility (sub-) requirements: (1) It should be flexible to enable a simplified and module-based customisation; (2) It should be generic rather than specific, i.e. mostly consist of essential BP elements supported by the majority of BP representations in order to allow for bi-directional transformations to and from these representations. The generic nature is also important considering the prevalent gap in abstraction (see Section 3.1) and the challenges of process discovery (see Section 2.4.1); (3) It should be a holistic language covering all features (qualitative and quantitative) of the observed system. That means in particular the sufficient representation of all relevant BP perspectives and the support for complex structures such as sub-processes, multiple parallel processes acquiring the same resource pools, etc. (see left in Figure 4.6 where the control-flow perspective is represented by the concept of a control-flow scenario which in turn contains a number of dependent or independent (sub-) processes).

It can be argued, that the variability requirement from the previous section is a logical subset of the expressibility requirement since flexibility by design via decisions, etc. is an

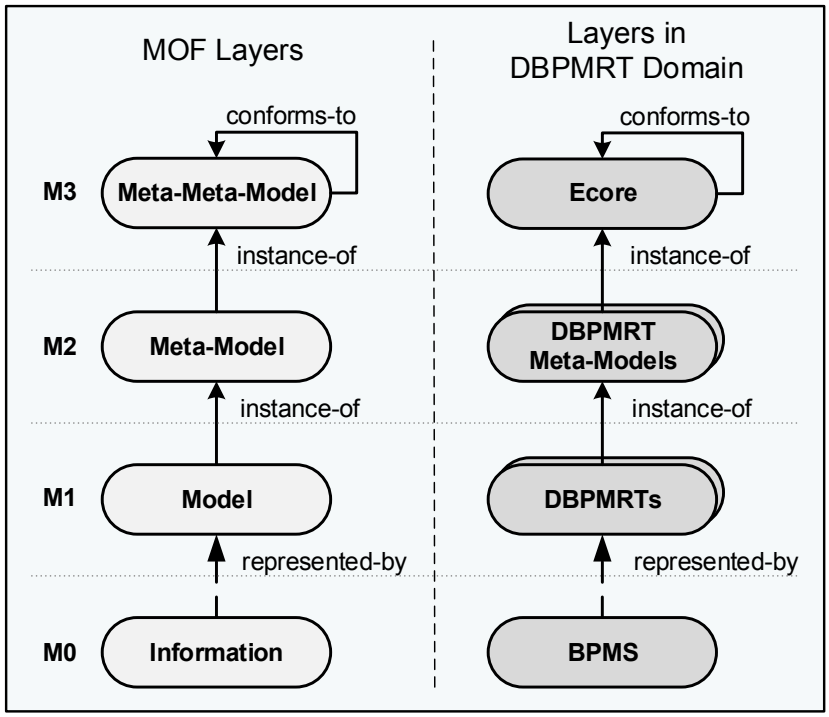


Fig. 4.7 Model Layers of the DBPMRT in comparison to the MOF Layers [162]

expressibility feature of all prominent BP languages/standards. Considering this observation the requirements for a DBPMRT can be summarised in a simplified way as follows: *A DBPMRT should be able to capture information on the instance as well as the type level while at the same time support expressibility features specific to the BP domain.*

4.2 Descriptive Business Process State Model at Run-time

One of the two types of DBPMRT is a state model, i.e. the DBPMRT reflects the current run-time state (on type and instance level) of the BPMSs. It consists of the three perspectives of control-flow, resources, and performance plus the information about the current state of the traces. The current state of the business process acts as an input for known BPM reasoning techniques like simulation in order to perform for instance a prediction or what-if analysis. The main challenge for the DBPMRT state language is to find a generalised state representation that can be analysed with existing reasoning methods and at the same time supports the common element types that can be found in popular business process standards.

In this section the meta-models that are making up the state DBPMRT are presented. The meta-models and models are designed, maintained, and managed using tools of the Eclipse Modeling Framework (EMF) [225], i.e. the meta-models conform to the *Ecore* meta-meta-model and represent the language the DBPMRT instances conform to. The model layers in the DBPMRT domain are shown in Figure 4.7 in relation to the MOF Layers [162]. The individual DBPMRT meta-models are presented in the following:

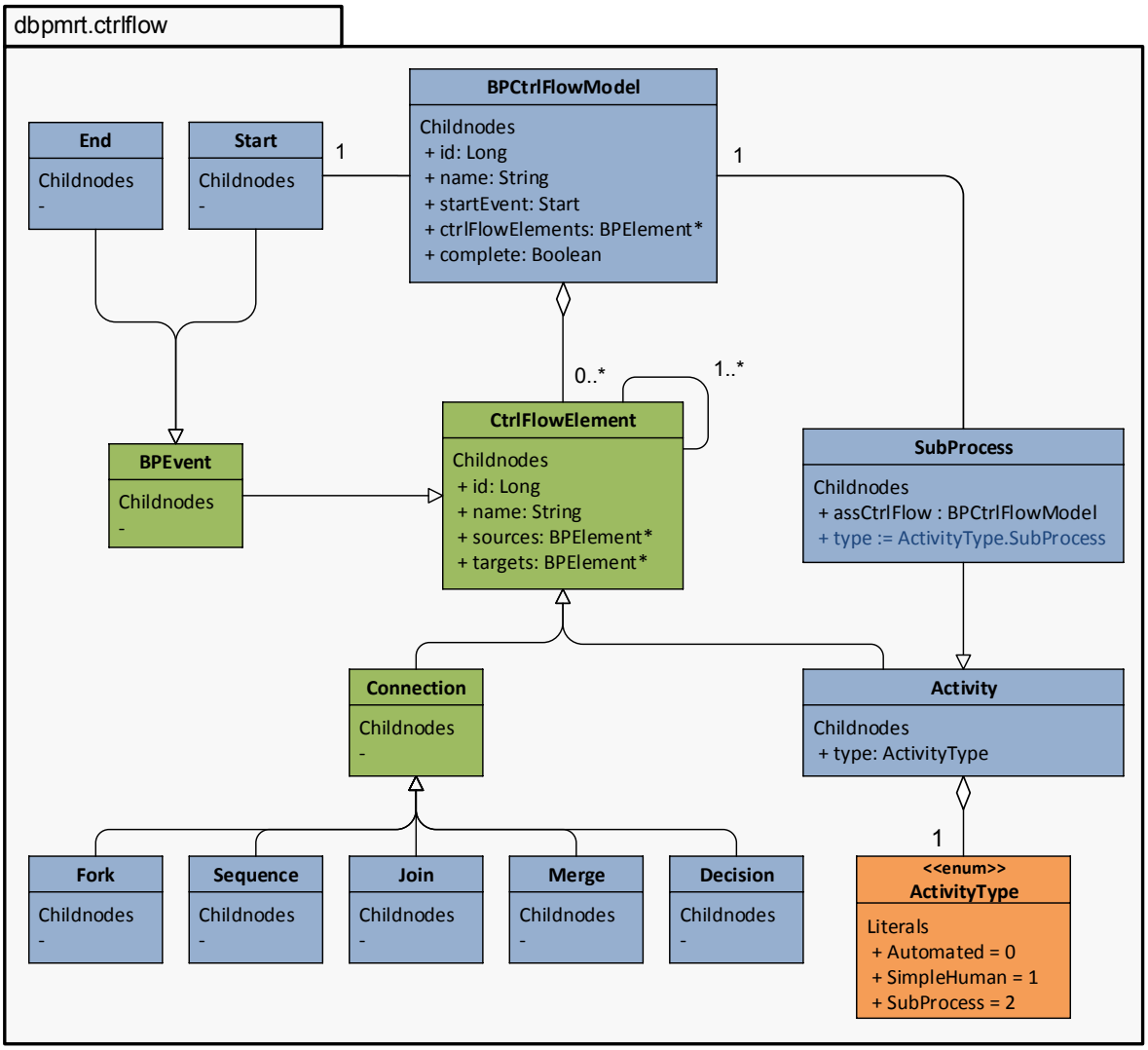


Fig. 4.8 Meta-Model for Single Control-Flow Models in State DBPMRT

4.2.1 Control-Flow Perspective

The control-flow is considered to be the core part of a business process and as such other perspectives are often based on elements of the control-flow, e.g. a role is associated with a set of activities. Figure 4.8 shows the meta-model from which a specific control-flow can be instantiated. Note, that abstract (non-instantiable) classes/interfaces are depicted in green, instantiable classes in blue, and enumerations in orange. A few important specifications of the meta-model are highlighted in the following:

- The parent element to which all other elements that are part of this control-flow have a transitive containment relation[2] is the *BPCtrlFlowModel*. An instance of the *BPCtrlFlowModel* has an `id`, a `name`, contains `ctrlFlowElements`, and has a refer-

[2] A diamond shape at the beginning of the connection determines that the target is logically a part of the source element, i.e. the target is contained in the source; An element must have exactly one direct containment relation to a parent with the exception of the top-level parent element, e.g. the *BPCtrlFlowModel* in the case of the control-flow. An arrow-less and diamond-less connection constitutes an association between elements without containment relation, i.e. a reference.

ence to exactly one `start` element (which is already contained as `ctrlFlowElement`). Furthermore, it has a flag `complete` that indicates if the control-flow model is sound, i.e. is block-structured and does not possess group activities (later explained in Section 5.3.4).

- The core element class from which every other class (apart from the *BPCtrlFlowModel* and the *ActivityType*) is derived[3] is the *CtrlFlowElement*. All control-flow elements, e.g. *Start* and *End Events*, *Activities*, as well as *Decisions*, *Merges*, *Forks*, and *Joins* (see Section 2.1) are a specialisation of the *CtrlFlowElement* which has an `id`, `name`, and source and target references to other *CtrlFlowElements* (fields `sources` and `targets`). The number of permitted sources and targets is dependent on the eventual specialisation, e.g, a *Decision* has exactly one source and two or more targets, a *Merge* two or more sources and exactly one target, a *Start* no source and exactly one target, and an *Activity* exactly one each, etc.

- There are three different types of activities (see Section 2.1): Automated, Human, and Sub-Process. While automated, and human activities are instantiations of the *Activity* class (defined by the respective `type` field) the *SubProcess* is a specialisation of the *Activity* class (with the `type` field enforced to `ActivityType.SubProcess`) and an additional reference to another *BPCtrlFlowModel* (outside of the transitive containment relation of the *SubProcess's* control-flow). Note, that with the presented meta-model only single BP control-flows can be instantiated. A collection of connected (via *SubProcess*) or independent control-flows in order to fully reflect the systems control-flow perspective is presented in the Section 4.2.5 introducing the meta-model for a holistic BP model.

4.2.2 Resource Perspective

Figure 4.9 shows the meta-model that specifies the resources perspective. The resource perspective contains information about which *Resource* is associated to which *Role(s)* and which set of *Activities*[4] is associated with a specified *Role*. The parent element containing all *Resources* and *Roles* is the *BPResourceModel*. Both, *Roles* and *Resources* have `names` and unique `ids`, respectively. Technically, two types of resources exist: (1) automated actors which are non-blocking (one resource may process multiple activities in parallel without influence on the respective processing times (no queue), e.g. a cloud web-service) and can only be associated to roles representing automated activities; (2) human actors associated to roles representing human activities (blocking; with queue). The flag `automated` specifies the resource type: `true` representing the former type and `false` the latter.

[3] Connections labelled with an arrow represents a specialisation relation.

[4] Elements of the language defined in other meta-models, e.g. *Activity* defined in the *dbpmrt.ctrlflow* meta-model, have have a purple colour.

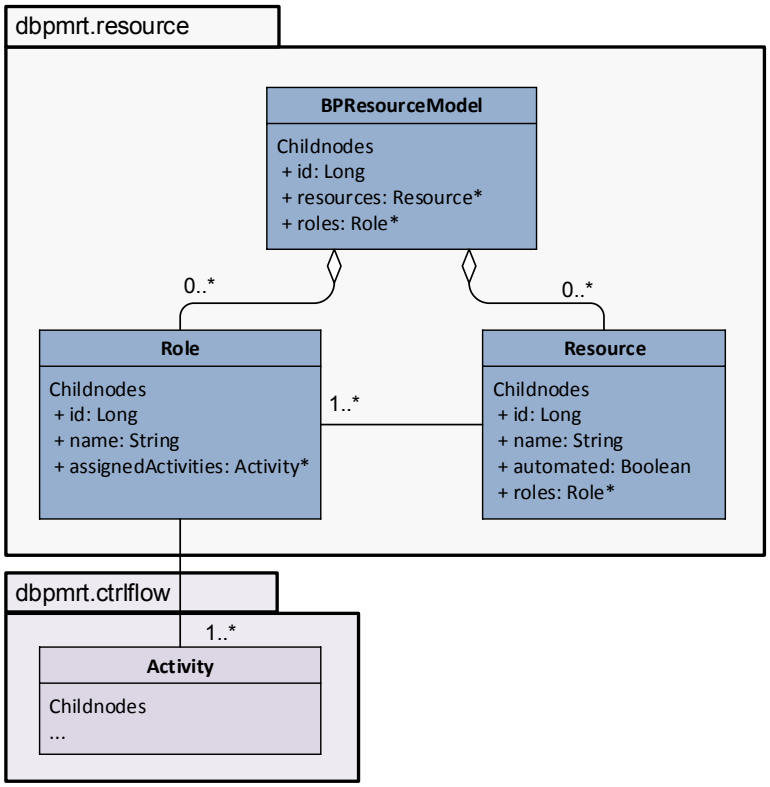


Fig. 4.9 Meta-Model for Resource Perspective in State DBPMRT

4.2.3 Performance Perspective

Figure 4.10 depicts the meta-model for the performance perspective of the DBPMRT. The parent element to which other elements of this meta-model (in the *dbpmrt.performance* package) have a containment relation is the *BPPerformanceModel*. The meta-model only shows three selected performance parameters (see PPIs in Section 2.5.1): process instance occurrence (*ProcessInstantiation* in the meta-model), activity net working time (*ActivityWorkload*), path probabilities (*BranchingProbability*). These three PPIs are in fact not reflecting the internal performance of the process/system but rather describe external environmental circumstances, i.e. they are influenced from outside of the business process. For instance, the process instance occurrence of the online ordering process in Figure 2.2 (page 14) is driven by demand, season, and other factors but not (directly) by how well the process performs. These kind of PPIs are also referred to as *external PPIs* and are required to carry out performance prediction reasoning via simulation (see Section 2.5.2), the reasoning use case examined in the context of this thesis. Generally, the PPIs contained in the performance perspective of the DBPMRT are aligned with the desired reasoning purpose. If required, other PPIs, such as *Resource Utilisation* and *End-to-End Processing Time*, can be included in a similar way as the ones depicted in the meta-model.

Each of the three PPIs in the model has a reference to the respective element that the performance describes, i.e. *ProcessInstantiation* with a *Start*, *ActivityWorkload* with an *Activity*, and *BranchingProbability* with a *Decision*, and contains a representation of the respective performance value, i.e. *ProcessInstantiation* and *ActivityWorkload* con-

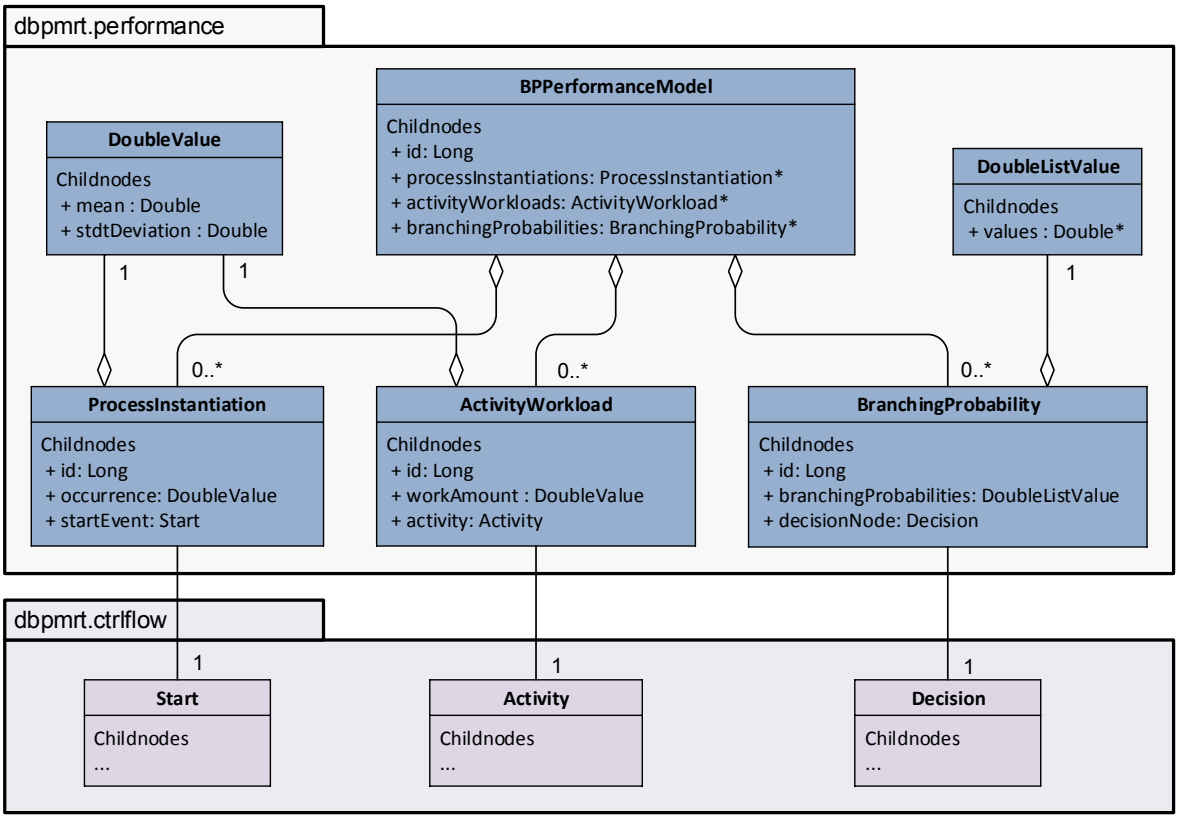


Fig. 4.10 Meta-Model for Performance Perspective in State DBPMRT

tain a *DoubleValue* representing `occurrence` and `workAmount`, respectively, and *BranchingProbability* contains a *DoubleListValue* representing the `branchingProbabilities`. *DoubleValue* and *DoubleListValue* are aggregated type level performance values:

- A *DoubleValue* is represented by the average aggregated over multiple instances (field `mean`) and a measure that indicates by how much the actual values of the instances deviate from the average, i.e. the standard deviation (field `stdDeviation`). Both values form a *Normal Distribution* [104] which serves as a common representative in the probability theory to generalise/aggregate from a sequence of parameters. If only the mean of the PPI is to be recorded the standard deviation (default) value is 0.

- A *DoubleListValue* already represents a distribution value, albeit for a discrete rather than a continuous domain (i.e. no standard deviation). In the meta-model it is used to describe the respective probabilities of the different outgoing paths of a decision. The order of the probabilities in the `values` list has to correspond to the order of the outgoing paths in the `targets` list of the *Decision*, e.g. the first value describes the probability of the first outgoing path. Furthermore, it is assumed that the sum of the probabilities in the list equals 1.

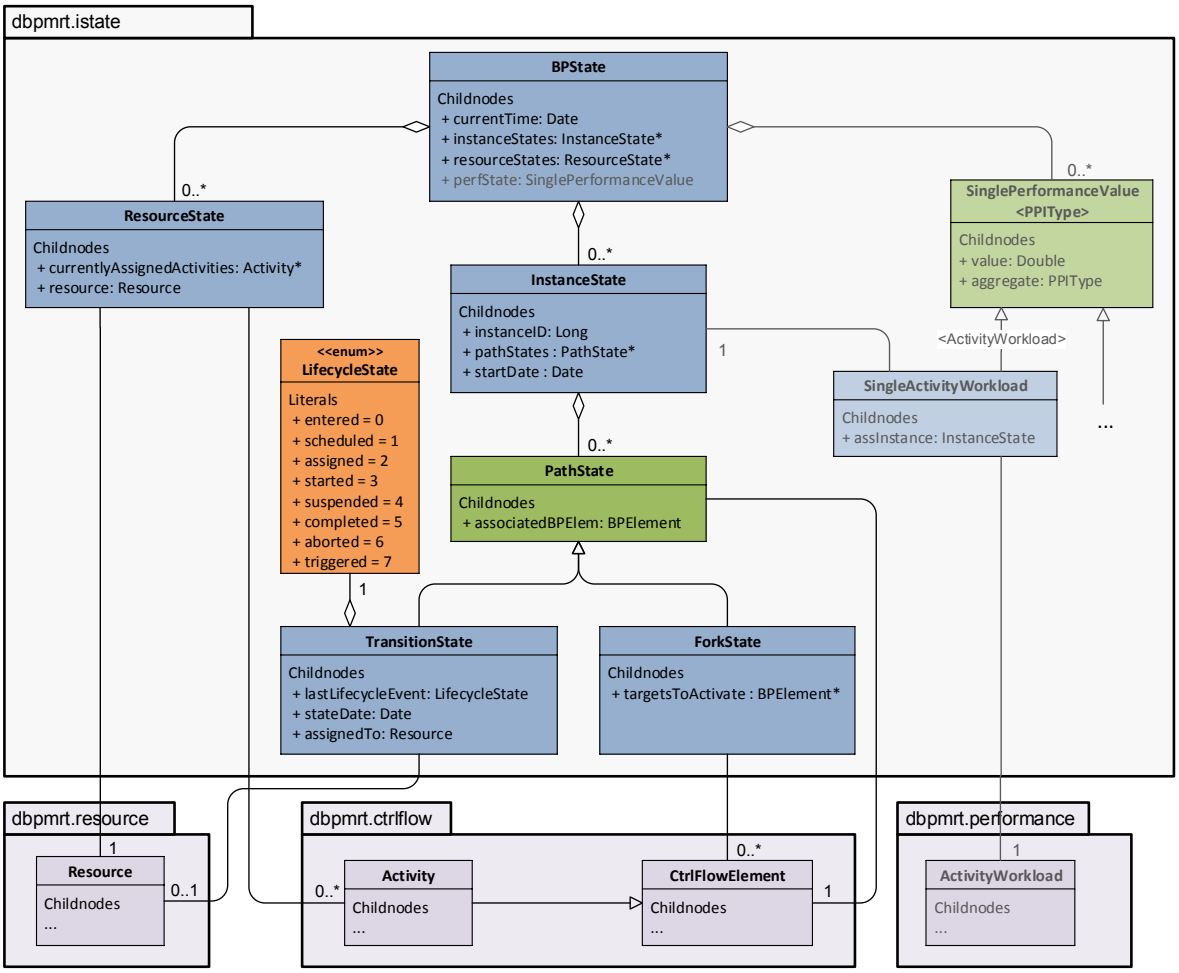


Fig. 4.11 Meta-Model for Process Instance Perspective in State DBPMRT

4.2.4 Process Instance Perspective

While in the previous sections the meta-models for type level perspectives control-flow, resources, and performance were proposed, this section is concerned with meta-modelling the instance level of the DBPMRT. Figure 4.11 shows the proposed meta-model. A few features are highlighted in the following:

- The parent element to which all other instance state elements have a transitive containment relation is the *BPState*. Basically, internal fine-granular states can be associated with either resources or BP instances. The states (if captured) of other more collection-based elements such as *Role* are aggregates of these fine-granular states.
- The state of a resource is captured as *ResourceState*. It has a reference to the *Resource* it is representing and references to all *Activities* to which the resource is currently assigned. If the resource is `automated`, more than one activity might be associated to its state; otherwise at most one.
- A *InstanceState* represents the internal state of an actively processed BP instance. It mainly describes the current state in the control-flow (via `pathStates`) but may

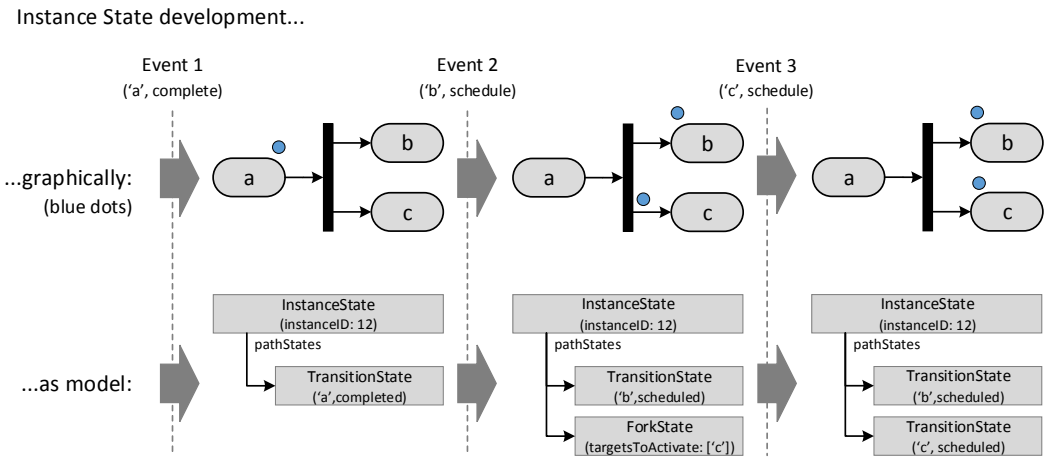


Fig. 4.12 The Development of the Instance State (Model) for an Example Stream of Events

potentially also contain information about performance or data aspects. Since the data perspective is highly system- and domain-specific it cannot be supported by a generic solution as proposed in this thesis. Hence, it is not part of the meta-model.

- The instance level performance state is the single occurrence of a PPI (the aggregated value as meta-modelled in the performance perspective in Figure 4.10). The reference meta-model includes as an example the instance level performance state of the *ActivityWorkload* PPI (greyed out elements on the right of Figure 4.11): *SinglePerformanceValue* is an abstract class generic to <*PPIType*> that contains the recorded `value` of the performance parameter PPIType (see `aggregation` field). *SingleActivityWorkload* is an example of an realisation/specialisation of the *SinglePerformanceValue* where the *PPIType* is resolved to *ActivityWorkload*, i.e. the reference `aggregate` is of the type *ActivityWorkload*. Additionally, it has a reference to the *InstanceState* for which the PPI was recorded. Other PPIs, e.g. process instance occurrence, are meta-modelled in the same way as specialisation of the abstract class `SinglePerformanceValue` but may have references to other state elements, e.g. *ResourceState* if the PPI is resource-related.

- The state of the control-flow for a BP instance is reflected by its *PathStates*. An *InstanceState* can have more than one current *PathStates* due to parallel constructs, i.e. state of path p_1 plus state of path p_2, etc. Possible realisations of a *PathState* are a *TransitionState* and a *ForkState*. A *TransitionState* reflects the actual events and has a reference to the corresponding *CtrlFlowElement* and a *Resource* if available. A *ForkState* on the other hand is the result of entering a fork (see Figure 4.12): When an event associated to one of the target paths occurs (see Event 2), a *ForkState* represents that other paths still have to be activated (the lower path including activity 'c' in the example case). This is resolved when an event associated with the "still to activate" path occurs (see Event 3).

4.2.5 Holistic DBPMRT

The holistic DBPMRT for states is the combination of the previously presented perspectives with the addition to be able to reflect systems that employ more than one BP control-flow. In Figure 4.13 the meta-model for the holistic state DBPMRT is depicted: It consists of an `id` field and directly contains the resource, performance, and instance state perspectives. Furthermore, it contains a *BPScenarioModel* which acts as a container for all control-flows (some inter-connected via the sub-process relation) that are captured from the run-time system. In particular, a *BPScenarioModel* has an `id`, a `name`, contains the respective control-flow models (`ctrlFlows`), and has a reference to the control-flow models (contained via `ctrlFlows`) that are externally instantiated, i.e. not internally as sub-process (`instantiatableCtrlFlows`; it is a subset of `ctrlFlows`). If we consider the Online-Order-Processing BP from Figure 2.2 (page 14): While the main process is an instantiable control-flow, *Process Order* (the grey activity) is a sub-process and only triggers system-internally, i.e. *Process Order* is not in `instantiatableCtrlFlows`. Note, that being a sub-process and being externally instantiatable is not mutually exclusive, i.e. a control-flow can be both.

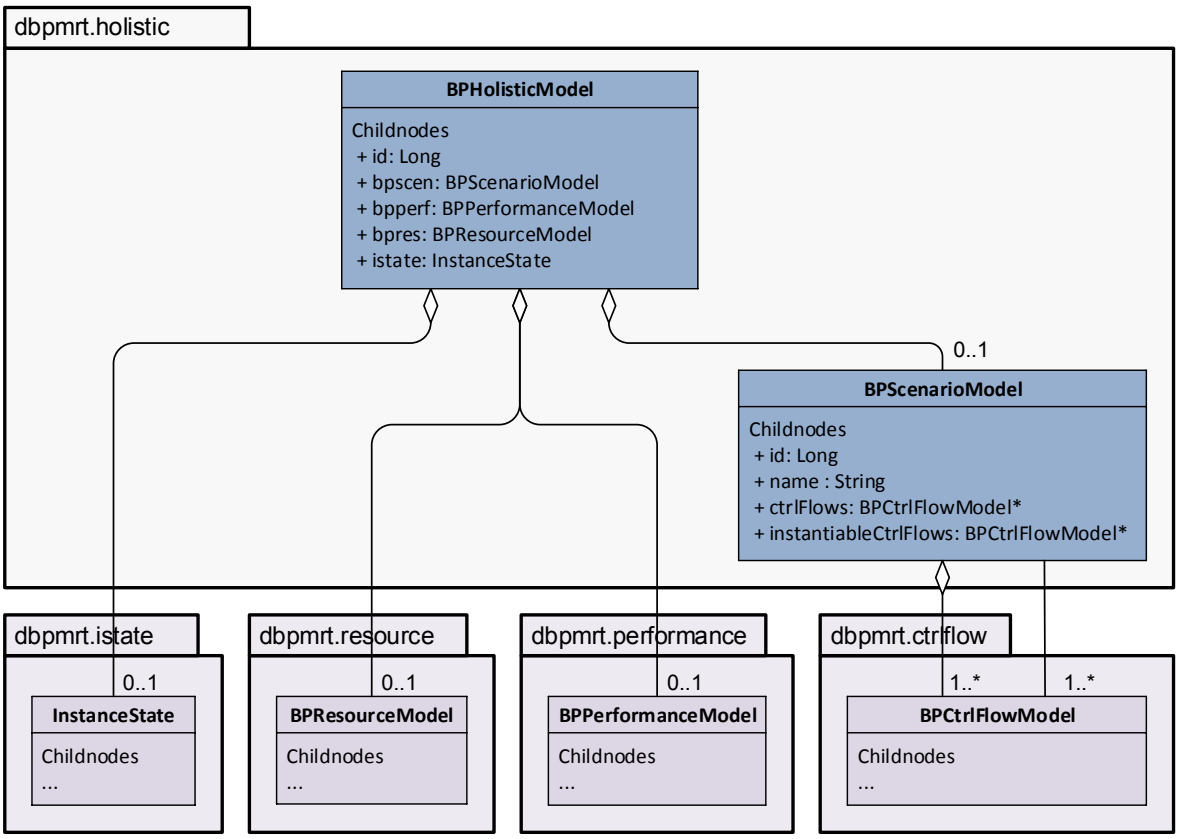


Fig. 4.13 Meta-Model for the Holistic State DBPMRT

4.3 Descriptive Business Business Process Evolution Model

The second type of the DBPMRT is an evolution model. Different to the State DBPMRT proposed in the previous section, the Evolution DBPMRT allows a stronger support for the dynamism requirement, i.e. while the state DBPMRT only allows for changes on the type level (i.e. updating information) the evolution DBPMRT records all states (i.e. adding information). This means, in practical terms, that while the former just captures/updates the current state, the latter introduces an additional time dimension that allows to map states to a specified time values (dates or timespans) and does not discard old states (similar to the "Big Data" principle). However, this is restricted to aggregated type-level information, i.e. the control-flow, resource, and performance perspectives. The evolution of the instance state is not required (and sufficient) to be stored since it is already available as abstracted data: the aforementioned type level perspectives. The DBPMRT as evolution model proposed in this section essentially acts as input for further high-level reasoning such as discovering change or adaptation patterns.

4.3.1 Structure of Evolution DBPMRT

The type level perspectives of the State DBPMRT as established in the previous section is the basis for the Evolution DBPMRT. Including the time dimension and thus making relations between model elements temporal is a non-trivial task and provides many different solutions which depend on the characteristics of the (meta-)model and, even more importantly on the frequency of change of individual elements. If the model structure would be solely hierarchically similar (i.e. tree structure with only containment relationships - containments are the primary order relations here) without any further interconnections (references are secondary order relations) the problem could be solved by replacing the containment relation with a temporal containment relation (discussed later). This is not the case with BP models in general and the DBPMRT in particular as shown in Figure 4.14: Here, key elements of the state DBPMRT are depicted along with their hierarchy level (dark shade = top level; light shade = low level) defined by the containment relations between them (diamond shaped arrows). Note, that all basic control-flow elements, e.g. activity, decision, etc., are conflated to the representative *BPCtrlFlowElement*. Additionally, important references between the elements (simple arrows) are shown adding to the complex structure of the DBPMRT. This makes the introduction of temporal relations into the DBPMRT more difficult.

One naïve solution to transform the DBPMRT into an evolution model would be to introduce a new parent element containing a list of *BPHolsiticModels* each associated with their respective time of relevance. This solution would require to recreate a new complete holistic model for each time a change occurred. Especially the changes in the performance perspective are only small but very frequent which makes this solution relatively unsuitable. A more fine-tuned option is to take the frequencies of certain changes into

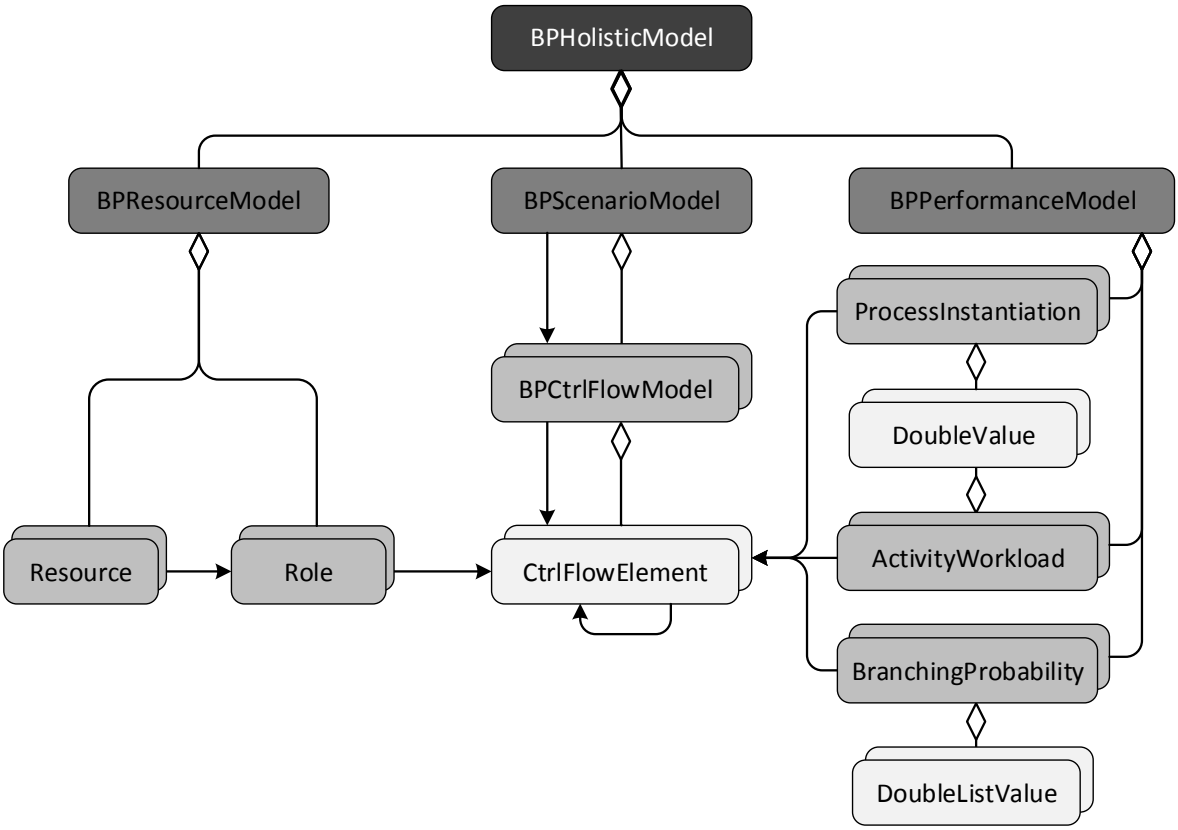


Fig. 4.14 Hierarchy (via Containment Relation) of a DBPMRT and important Associations between Elements

account and strategically replace containment or reference relations with their temporal equivalents (i.e. relations annotated with their time of relevance). One such possibility is to define "spheres of change" which correspond with the different perspectives, e.g. if something in the resource perspective changes, a new time-associated *BPResourceModel* is added. Figure 4.15 shows an example of this approach: The different colours indicated the different "spheres of change" and whenever relations are crossing these spheres they are required to become an equivalent temporal relation, i.e. *temporal containment* and *temporal reference*. The realisation of these relations is discussed in Section 4.3.2. Other relations[5], not crossing spheres, remain as before. Note, that the spheres are not in every case matching the separated perspectives, e.g. the sphere for control-flow changes reaches into the performance perspective. These deviations are the result of considering the special characteristics of BP changes:

- Since the performance perspective changes with a very high frequency relative to the control-flow perspective it is not feasible for each such change to create a complete new *BPPerformanceModel* including all PPIs (*ProcessInstantiation, ActivityWorkload*, and *BranchingProbability*) with all its references to the control-flow. Furthermore, the respective PPIs are directly connected to *BPCtrlFlowElements* and

[5]Figure 4.15 shows is a simplified view of the DBPMRT. In the complete model are more relations that are not crossing spheres.

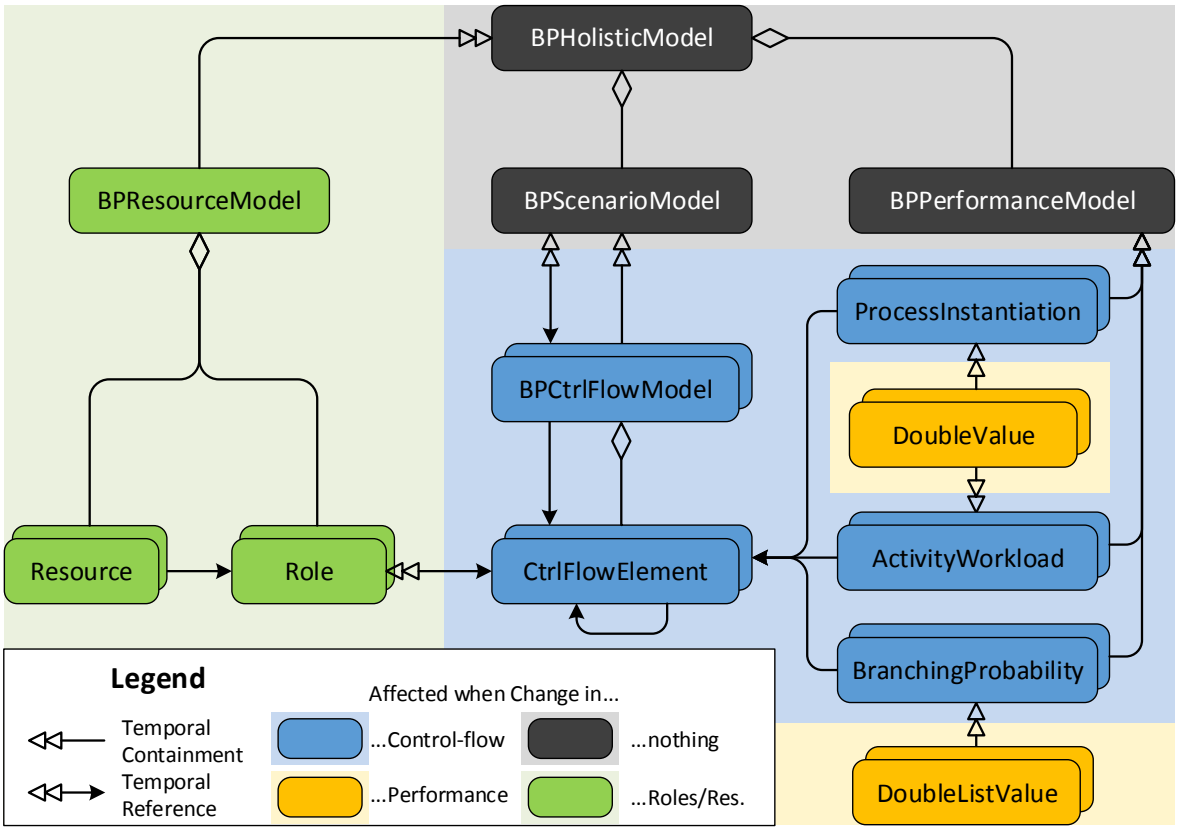


Fig. 4.15 Influence Spheres of Certain Changes and the Situation of Temporal Relations in the Evolution DBPMRT

thus fall logically rather into the change sphere of the control-flow perspective. As a result of the inclusion of the PPIs into the control-flow sphere of change, a changed performance is reflected by the temporal relation to each PPI's value, i.e. *DoubleValue* or *DoubleListValue*.

- The DBPMRT captures the state and evolution of multiple control-flows (see *BPScenarioModel*). Usually if one control-flow changes, the others are unaffected by this. This is the reason to differentiate between changes of individual control-flows and not conflate them with each other. Hence, the introduction of the temporal relations between *BPScenarioModel* and *BPCtrlFlowModel* rather than between *BPHolisticModel* and *BPScenarioModel*.

If a system features extremely volatile resource or control-flow models it can make sense to position the temporal relations on an even lower level. Possible candidates are: (1) roles and resources, e.g. if the resources change very frequently but the roles stay mostly the same it can make sense to not create the entire *BPResourceModel* new for each resource change but instead decouple them, i.e. make all relations in the *BPResourceModel* temporal; (2) activities and other control-flow elements, i.e. the set of activities in a control-flow does change very rarely in comparison to the general control-flow (execution order). In this scenario it can make sense to separate an activity sphere of change from the control-flow sphere of change.

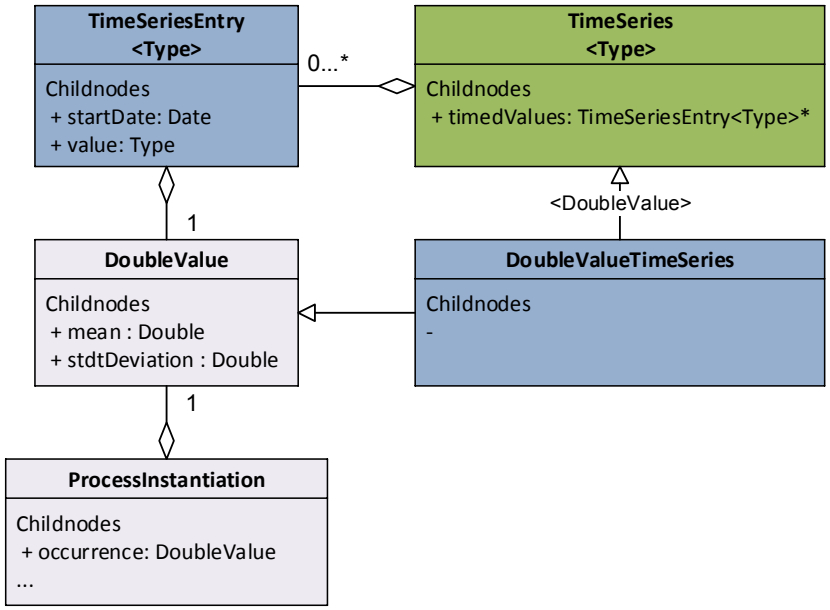


Fig. 4.16 Model Schema for Temporal 1:1-Containment for *ProcessInstantiation-DoubleValue* Relation

4.3.2 Temporal Relations for Evolution DBPMRT

The schematic DBPMRT as evolution model was introduced in the previous section containing 10 different temporal relations. In essence they conform to three different types that need to be meta-modelled. The realisations of these three types are described in the following, each exemplary on the basis of a reference temporal relation.

Temporal 1:1-Containment Relation

One of the types is a temporal 1:1-containment relation, i.e. the parent element contains one child element. The relation between the PPI *ProcessInstantiation* and *DoubleValue* is such a relation. Figure 4.16 depicts the proposed meta-model to realise the temporal relation for this type: Two new key classes are introduced, *TimeSeries* and *TimeSeriesEntry*. The latter is a typed element (<*Type*>) that contains the typed `value` and a `startDate` representing the start time of relevance of the `value`. The former is a typed abstract class that contains *TimeSeriesEntries* of the same type. With the help of these two basic classes the temporal relation is realised by the new class *DoubleValueTimeSeries* which is a specialisation of both, the *DoubleValue* and the typed *TimeSeries<DoubleValue>*[6]. Through the resolved type being *DoubleValue* all *TimeSeriesEntries* do now contain a `value` of type *DoubleValue*. With regards to the classes *ProcessInstantiation* and *DoubleValue* nothing in the original meta-model definition changes; only new elements are included and allow for using the temporal (*DoubleValueTimeSeries*) rather than the state (*DoubleValue*) element for the `occurrence` value. Through the proposed realisation *ProcessInstantiation-DoubleValue* essentially becomes a 1:*-relation since many values are now contained

[6] EMF allows multiple super types, i.e. multiple inheritance

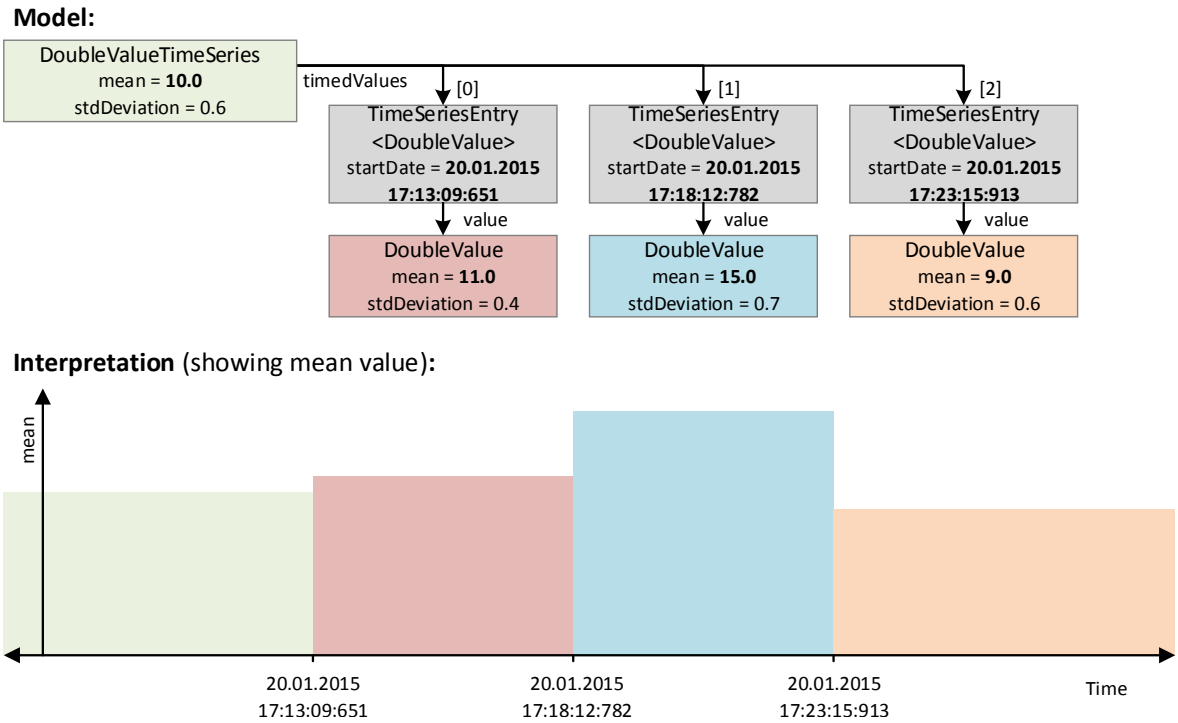


Fig. 4.17 Example Model for a *DoubleValueTimeSeries* Temporal Relation and its Interpretation

in this PPI. However, given a specified time/date the relation stays of the type 1:1 as intended.

An example model of such a temporal relation (*DoubleValueTimeSeries*) and its interpretation is shown in Figure 4.17: The values belonging to the *DoubleValueTimeSeries* (it is a specialisation of *DoubleValue* and thus also has `mean` and `stdDeviation`) are associated with the time before the `startDate` of the first entry in the `timedValues` (light green colour). The *DoubleValue* of the first *TimeSeriesEntry* is valid from its `startDate` to the `startDate` of the next entry. If there is no next *TimeSeriesEntry*, the associated *DoubleValue* is valid from *startDate* to infinity.

Other relations of a similar type and which are realised in the same way are between *ActivityWorkload* and *DoubleValue*, *BranchingProbability* and *DoubleListValue*, as well as *BPHolisticModel* and *BPResourceModel*.

Temporal 1:*-Reference Relation

Another type is the temporal 1:*-reference relation as existent between *Role* and *Activity*[7]. Figure 4.18 shows its realisation. Again with the help of the two core classes *TimeSeries* and *TimeSeriesEntry* the *Role-Activity* is transformed into a temporal relation. For this, two new classes are created: *ActivityList* essentially representing the 1:* characteristic of the relation and *ActivityListTimeSeries*, a specialisation of *ActivityList* and *TimeSeries<ActivityList>* representing the temporal relation. However, the original meta-model

[7] One *Role* has references to multiple *Activities*, i.e. 1:* relation.

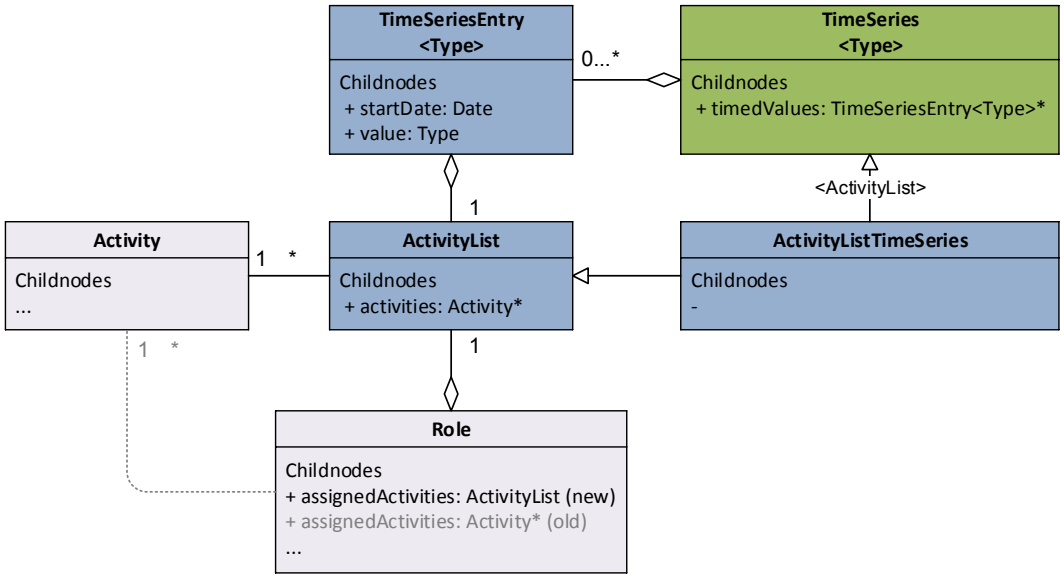


Fig. 4.18 Model Schema for Temporal *Role-Activity* Relation

had to be slightly altered: now `assignedActivities` in *Role* contains one *ActivityList* rather than multiple *Activities* directly. This alteration allows for both: the model to be used as state model (when `assignedActivities` is an *ActivityList*) and as evolution model (when `assignedActivities` is an *ActivityListTimeSeries*).

Temporal Grouped Relation

A third type used in the evolution DBPMRT is the temporal group relation which is a collection of relations from one source to a number of targets (possibly of the same type) that change at the same time, i.e. can be temporally grouped. One such group are the relations from *BPScenarioModel* to *BPCtrlFlowModel*. Figure 4.19 shows the meta-model extension for this relation. It is similarly realised as with the previous temporal relation: A proxy class is modelled that groups the considered relations into one entity (*CtrlFlowRelations*) with references copying said relations (`ctrlFlows` and `instantiableCtrlFlows`). Furthermore, *BPScenarioModel* is amended so that it contains all *BPCtrlFlowModels* (`ctrlFlows`) and one proxy class *CtrlFlowRelations* (`temporalCtrlFlows`) but not the relations that have been moved to the proxy class. With the second new class *CtrlFlowRelationsTimeSeries* (a specialisation of *CtrlFlowRelations* and *TimeSeries*<*CtrlFlowRelations*>) the meta-model allows for evolution modelling (when `temporalCtrlFlows` is an *CtrlFlowRelationsTimeSeries*) and state modelling (when `temporalCtrlFlows` is an *CtrlFlowRelations*).

The same pattern is also applied in the temporal group relation between the *BPPerformanceModel* and its respective *ProcessInstantiations*, *ActivityWorkloads*, and *BranchingProbabilities*.

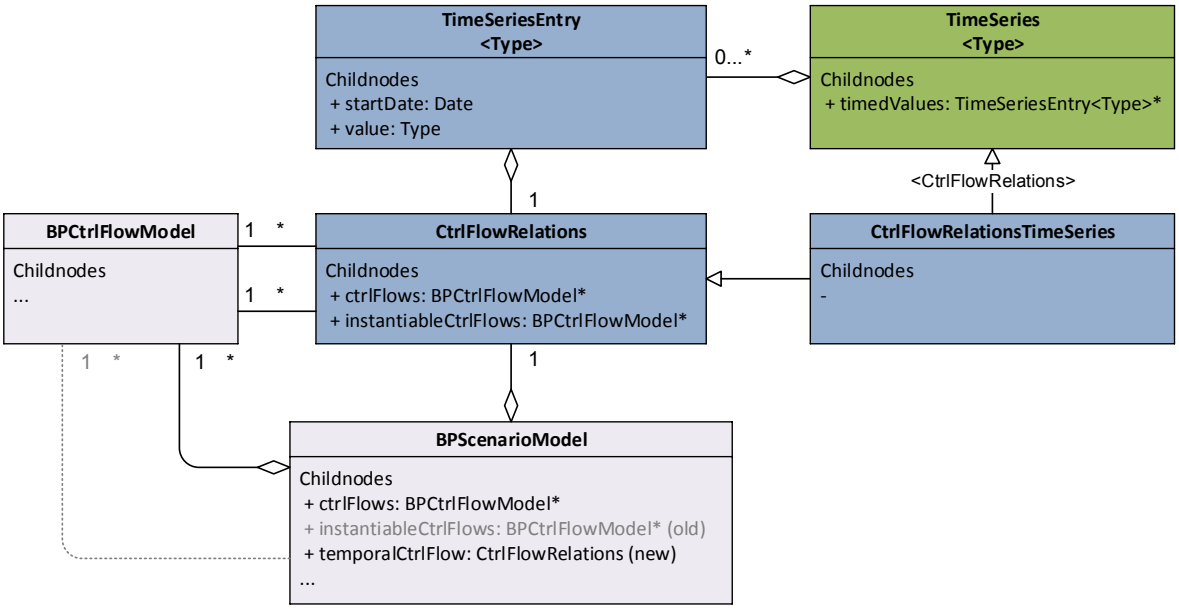


Fig. 4.19 Model Schema for Temporal *BPScenarioModel-BPCtrlFlowModel* Relation

4.4 Qualitative Evaluation

The evaluation of the proposed DBPMRT meta-models is twofold:

1. Primarily through an evaluation via a case study (see Chapter 6): There the proposed DBPMRT is *quantitatively evaluated* for its capability to represent a BP simulation model in Chapter 6 in the context of predictive performance reasoning. The use of case study for evaluation is a standard evaluation methodology to analyse the usefulness of model artefacts in software engineering (see Section 1.6).

2. Secondly, through a supplementary *qualitative evaluation* of the reference meta-models with regards to meeting the DBPMRT requirements MR1-MR4 established in Section 4.1.2. The goal is to evaluate whether the proposed reference language is expressive enough to represent domain-specific constructs while at the same time not allow for much redundancy.

The qualitative evaluation with regards to meeting the defined requirements is carried out in the remainder of this section:

Reflectivity (MR1): In order to meet the reflectivity requirement, modelling capabilities were introduced that allow the capturing of instance level information. This includes instance state information for the control-flow, resources, and performance perspective. The proposed DBPMRT language allows for the capturing of these types of instance level information (see Section 4.2.4). Since the DBPMRT is a use case independent solution, the proposed meta-model has been designed to generalise from the actual reality, i.e. no system-specific data is captured such as "value of order" or "credit card details". Albeit being instance level information that is potentially useful for some form of reasoning,

this kind of data is use case- and system-dependent, i.e. it can not be easily included in a general-purpose DBPMRT. Furthermore, the evolution of the reflective state is not captured as this would quickly exhaust any amount of memory for very frequent BPs (e.g. 1000s of instance state changes per second).

Variability (MR2): Just like every BP language that allows for individual instantiation and flexibility on the instance level, the variability requirement is fully met by the proposed DBPMRT. This is achieved by including control-flow elements that support flexibility by design like *Fork, Join, Decision*, and *Merge* into the language (see Section 4.2.1). This means with regards to descriptive models that the instance information is sufficiently generalised and represented by the type level information, e.g. the control-flow is aggregated information inferred from sequences of BP instance states or the PPIs are aggregated information inferred from occurred single performance values. This narrative is an alteration of that of design models in MDE where model information on the instance level conforms to the type information.

Dynamism (MR3): Dynamism is the ability to represent change on the type level of the process. To fulfil this requirement two different types of DBPMRT have been proposed: (1) the *State DBPMRT* (see Section 4.2) which allows for changes on the type level but only captures the current as-is state and discards outdated information, and (2) the *Evolution DBPMRT* (see Section 4.3) that records the evolution of the type information, i.e. certain relations in the *State DBPMRT* are extended by a time dimension in order to retain information about historic states of the BP. While both models allow for changes on the type level, the evolution DBPMRT even allows for recording different variants and their respective time of validity.

Expressibility (M4): The expressibility requirement is the ability to model relevant information specific to the BP domain (this technically includes the variability requirement). To support this ability the proposed DBPMRT has the following characteristics: (1) the DBPMRT is modularly structured (see Section 4.2.5), i.e. allowing the modelling of basic BP structures and supporting extensions where necessary (e.g. more PPIs in the performance perspective), (2) it is of generic nature, i.e. all essential BP elements and their associations are supported, and (3) the DBPMRT is a holistic approach, i.e. all relevant BP perspectives are supported and it is possible to model complex behaviour or circumstances such as sub-processes or different processes accessing the same resource pool. However, the proposed DBPMRT is a generic solution, i.e. it may, in some cases, be considered too much of a simplification/generalisation of the real system. For instance, resources have different efficiencies or other statuses, e.g. on holidays, sick, etc. Such missing modelling capabilities can be added to the meta-models if required. The aspect of possible oversimplification will be further discussed in the evaluation of the overall and reasoning framework in Section 6.3.3.

Conclusion: Based on the results of the qualitative analysis the author has assigned a semi-quantitative measure representing the degree of fulfilment achieved by the presented DBPMRT for each of the model requirements. The result is shown in Table 4.1.

Table 4.1 Degree of Requirement Fulfilment

(++ = Full, + = Mainly, ∘ = Partly, − = Negligible)

Requirement:	Principal Fulfilment	Restrictions	**Degree of Fulfilment**
MR1	Yes	- no system-specific data - no instance evolution	+
MR2	Yes	-	++
MR3	Yes	-	++
MR4	Yes	- simplification of real system	+

Aside from the concrete requirements for DBPMRT, the reference models can be evaluated against the identified gaps in the analysis carried out in Section 3.2. When regarding the individual features of the gap analysis (control-flow, organisational, and performance abstractions, as well as their state, and history) the proposed DBPMRT reference models represent a language that supports all theses features. This is in contrast to the state-of-the-art languages investigated in the gap analysis as shown in Table 4.2.

Table 4.2 Gap Analysis Descriptive Business Process Models at Run-time for DBPMRT; Legend: (-) not supported, (∘) somewhat supported, (+) mainly supported

BP Language	BP Type Perspectives			BP Instance State			BP History		
	Ctrl-Fl.	Org.	Perf.	Ctrl-Fl.	Org.	Perf.	Ctrl-Fl.	Org.	Perf.
EPC	+	+	+	∘	∘	−	∘	∘	−
BPMN	+	+	−	−	−	−	−	−	−
BPMN+	+	+	+	−	−	+	−	−	+
PN	∘	∘	∘	+	∘	−	−	−	−
HN	∘	∘	∘	−	−	−	−	−	−
YAWL	+	+	−	+	∘	−	−	−	−
ADEPT$_{flex}$	+	−	−	+	∘	−	+	−	−
DBPMRT	+	+	+	+	+	+	+	+	+

4.5 Summary

In this chapter the first objective of this thesis was addressed: The specification of a reference language for DBPMRTs. To achieve this, concrete requirements were formulated that need to be met by a modelling language for DBPMRT (see Section 4.1.2): capturing change on the levels of *dynamism*, *variability*, and *reflectivity*, as well as support the *expressibility* of common BP-domain models. To meet this requirements two languages were proposed: (1) reference meta-models for a DBPMRT were presented that are able to represent the holistic state (instance and type level + control-flow, resource, and performance perspective) of one or more observed BPMS (see Section 4.2), and (2) These reference meta-models were enhanced/adapted to enable the capturing of the temporal evolution of the BPMS's state on the type level (see Section 4.3). In the final Section 4.4, the DBPMRT specification was evaluated in a qualitative fashion against the requirements established earlier: It was shown that all requirements are at least mainly fulfilled with only minor restrictions that are subject of future work (see Section 7.2).

Chapter 5

Establishment of Causal Connection

BPMSs produce BP events indicating a change of the system's internal state. They represent fine-granule state transitions and are on a different abstraction level than business processes. This chapter addresses the objective of inferring the DBPMRT from this low level event data in a real-time environment, i.e. every change in the system should be *causally* and *timely* reflected in the associated run-time representation comprising information of different abstraction levels. In other words, the objective can be described as *online discovery of a holistic BP model* and as such consolidates a multitude of related domains: model-driven engineering, business process management, and data mining. In particular, the research areas of models at run-time (see Section 2.7), process discovery (offline (2.4.2) and online (2.4.4)), concept drift (2.4.3), simulation BP models (2.5.3) are of relevance for this objective. This chapter provides details about the developed agents and algorithms which are formally presented, evaluated, and thus addresses the state-of-the-art gaps identified in Section 3.3.

To address this objective, the chapter is divided in several parts (see Figure 5.1): First, the objective is further specified in Section 5.1, which includes the discussion of preliminary definitions and the specification of concrete requirements. In a second part a unique approach to *establish* the causal connection between BPMS and the DBPMRT is introduced in Section 5.2: The Construct Competition Miner (CCM) which discovers the control-flow of a process via a BP event log. The third part of this chapter is concerned with the issue of *maintaining* the causal connection during run-time. Concretely, a framework has been designed processing each BP event individually in order to adapt the BP state information on the type and instance levels and for all relevant perspectives. The general concept and components of the framework are explained in Section 5.3. On the basis of this concept the CCM was modified (utilising its unique features) to monitor an BP event stream and dynamically discover the control-flow. The design and modifications which lead to the development of the Dynamic Construct Competition Miner (DCCM) are discussed in Section 5.3.4. The specific algorithmic details for all components involved in the dynamic inference of type level information of all relevant BP perspectives are then explained in the following two Sections 5.4 and 5.5. Subse-

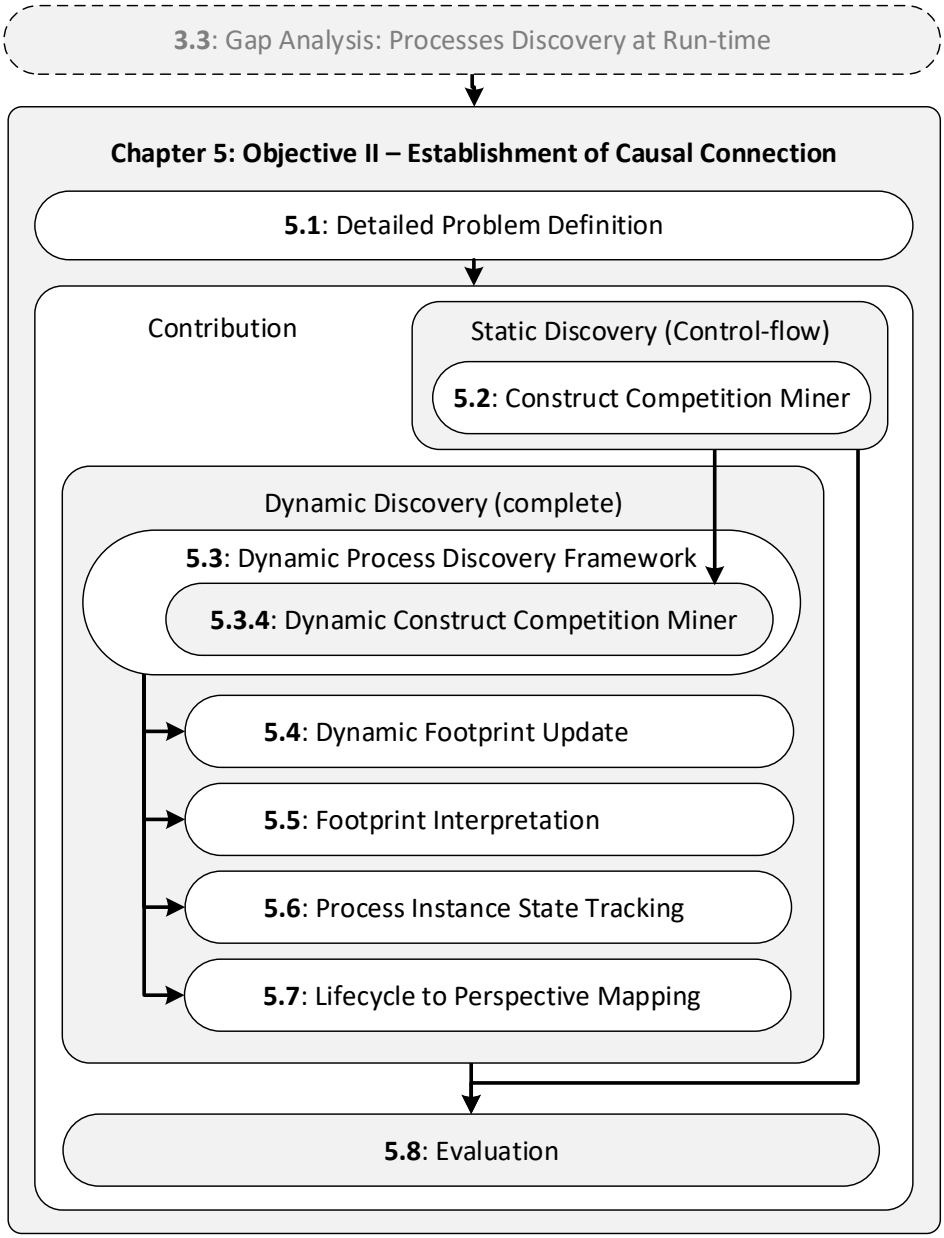


Fig. 5.1 Outline of Chapter 5

quently, in Section 5.6 it is discussed how to effectively track the instance state within the overall framework followed by an overview of how different event lifecycles contribute to the computation of the respective BP perspectives in Section 5.7. In a fourth part in Section 5.8, the algorithms concerned with the control-flow discovery, CCM and DCCM, are individually evaluated in a quantitative way via established quality criteria (see Section 2.4.1). Furthermore, a qualitative evaluation against the previously established requirements is carried out which includes all levels and perspectives. The chapter is summarised by a brief conclusion.

5.1 Detailed Problem Definition

The objective of establishing a link from BPMS to DBPMRT can be divided into two parts: (1) Establishing the causal connection and (2) Maintaining the causal connection in a dynamic real-time environment. The solution for both sub-objectives has to meet certain requirements which are explained in the remainder of this section.

Establishment of the Causal Connection

The abstraction gap between the events emitted by a BPMS and a BP model is difficult to bridge and only achievable with the help of complex aggregations. Bridging this gap of low level transactional information and a higher level model is the domain of contemporary data mining techniques. In the domain of business processes this is a challenging task in its own right highlighted by the fact that extensive research field (Process Mining/Process Discovery) has evolved featuring many substantial contributions as previously outlined in Section 2.4.2. The shortcomings of recent state-of-the-art algorithms are discussed in Section **??** and includes, for instance, the mining of Petri Nets instead of BP models, or not supporting noisy or incomplete event logs. Generally, traditional process discovery approaches are carried out in an static way as an "offline" method. This is reflected by the fact that the input for these algorithms is an entire log as expressed by the following definition:

Definition 1 *Let the log $L_n = [e_0, e_1, ..., e_n]$ be a sequence of $n+1$ events $e_0, e_1, ..., e_n \in E$ ordered by time of occurrence ($\forall\, i < j \wedge e_i, e_j \in L_n : time(e_i) \leq time(e_j)$) and BP_n be the business process model representing this sequence of $n+1$ events, then process discovery is defined as a function that maps a log L_n to a process BP_n, i.e.*

$$\begin{aligned} &ProcessDiscovery: \quad L_n \quad \Rightarrow BP_n \quad \text{or} \\ &ProcessDiscovery: [e_0, e_1, ..., e_n] \quad \Rightarrow BP_n \end{aligned} \tag{5.1}$$

In order to establish a causal link from BPMS events to DBPMRT through process discovery the following requirements[1] need to be met[2]:

TR1 *Independent/Autonomous:*
The enactment of BPs often deviates from the planned model which therefore does not properly reflect reality (see Section 3.1). This is why the discovery of a business process model from event logs should not rely on (possibly wrong or outdated) external a-priori information but instead act autonomously in order to extract an accurate as-is representation solely based on the reality captured in the log.

[1] TR1 and TR2 are derived from general process discovery requirements; TR3 is specific with regards to the problem of establishing a causal connection.
[2] TR = Tooling Requirement

TR2 *Well-fitting Result:*
The DBPMRT supports the most common elements and constructs of the existing standards of the business process domain. The algorithm is required to always discover such a generalised model even if the log contains sporadic, contradictory, or incomplete information, i.e. generalisation and simplicity are important quality criteria. But while supporting a certain degree of generalisation and simplicity, the agent should also achieve a high level of accuracy, i.e. precision and fitness, are similarly important to have an accurate reflection of reality. That means, the result should reflect the main behaviour while ignoring outliers (exceptional behaviour/noise), i.e. good balance between over-fitting and under-fitting (see Section 2.4.1).

TR3 *Robustness:*
Real life logs are likely to contain sporadic, contradictory, or incomplete behaviour. The agent has to *always* provide a (well-fitting) result independent of the input. Furthermore, a true causal link can only be established if the approach is a deterministic projection, i.e. the same input creates the same output.

The Causal Connection in a Real-time Environment

As elaborated in the state-of-the-art Sections 2.4.2 and 2.4.4 as well as the gap analysis Section 3.3, the challenges of establishing a causal link through traditional process discovery from event logs are generally motivated by algorithmic features (e.g. abstraction-based vs. heuristic, deterministic vs. non-deterministic) and their quality of the results (e.g. precision, simplicity, fitness; over-fitting vs. under-fitting). Of less or little importance on the other hand is the practical execution of these process discovery solutions for modern collaborative systems which represent a *large scale*, *high-frequency*, and *dynamic* environment. For such an environment it can be argued that just analysing a log in a static "offline" fashion can only *partly* claim to support a causal connection mainly due to (see Sections 2.4.4 and 3.3): (1) technical impracticability ("big data", i.e. data sets are too large and complex to store and process all of them), (2) technical delay (the log is already long out-of-date when the analysis has finished), and (3) the lack of provision for a BP's dynamic nature (most processes are not "steady-state" but instead evolve continuously).

In order to adequately respond to these new challenges the initial concept of process discovery has to be altered in order to allow for *dynamic process discovery* or *online process discovery* (see Section 2.4.4). Instead of the traditional static method (see Definition 1) dynamic process discovery is an iterative approach as defined in the following:

Definition 2 *Let the log* $L_n = [e_0, e_1, ...e_n]$ *be a sequence of* $n+1$ *events ordered by time of occurrence (* $\forall i < j \wedge e_i, e_j \in L_n : time(e_i) < time(e_j)$*) and* BP_n *be the business process model representing this sequence of* $n+1$ *events then dynamic process discovery is defined*

as a function that projects the tuple (e_n, BP_{n-1}) *to* BP_n*, i.e.*

$$DynamicProcessDiscovery: (e_n, BP_{n-1}) \Rightarrow BP_n \tag{5.2}$$

This definition entails that every run-time event (system transition) in the BPMSs is immediately processed according to the concept of CEP and BAM (see Section 2.2.3: Business Process Analysis in Real-time) and reflected in the continuously updated "current state" of the process. This includes potentially all relevant perspectives of a BP, i.e. control-flow, resource, performance, instance state. Taking the new challenges into account (dynamic, large-scale, high-frequency) and the adapted concept of dynamic process discovery, additional requirements should be met to maintain the causal link in this challenging environment:

TR4 *Detection of Change:*
While traditional process discovery approaches take into account the *Variability* level of change (see Section 4.1) a framework supporting dynamic process discovery should also detect change in the other two remaining levels to maintain the *causal* link: (1) *Reflectivity*: Change on the instance level, i.e. every single event leads to a fine-granular transition of the state of a trace, and (2) *Dynamism*: Change on the type level (Concept Drift), i.e. when a set of more recent events indicate a change of one of the type level perspectives of a business process this should be reflected in the respective parts of the output DBPMRT, e.g. because a trace appears that contradicts with the currently assumed control-flow.

TR5 *Optimised Algorithmic Run-time/Memory Usage:*
To fulfil the objective a *timely* reflection of causality is required. This aspect is for current control-flow discovery solutions of almost no importance. But since dynamic process discovery is adopting the event-by-event concept as specified in Definition 2 while potentially dealing with massive BPs consisting of hundreds of activities and producing thousands of events per second, the actual run-time of the algorithms becomes very important. As a consequence the event processing is required to be (1) quasi-immediate, i.e. each event needs to be processed within a small and near-constant amount of time, and (2) memory neutral, i.e. processing an infinite number of events without exceeding a certain memory threshold. Furthermore, it is desirable if the concept allows for scalability in order to cope with increasing workload at as little as possible additional computational cost, i.e. processing more events can be mitigated via scale-up or scale-out measures.

TR6 *Extensibility:*
As an MRT solution the framework should follow model-driven principles on the mega-model level and allow for customisation to suit specific input, output, and processing requirements, i.e.: (1) Since collaborative BPs often span multiple BPMSs, each potentially producing events of different formats, the concept is also required

to enable the introduction of additional adapters which allow for processing new event formats; (2) The concept is required to allow for individual processing components to be exchanged, removed, and/or added, dependent on the purpose and extent of the DBPMRT that is to be mined.

5.2 Static Construct Competition Miner

In this section the specifics of the Constructs Competition Miner (CCM) are described. The CCM was developed to meet requirements TR1 - TR3 and features unique characteristics that can be utilised to fulfil TR4 - TR6 (through modifications which are specified in Section 5.3.4). Generally, the CCM can be characterised as a deterministic process discovery algorithm that operates in a static fashion and follows a divide-and-conquer approach which, from a given event log, directly mines a block-structured process model that represents the main behaviour of the process. Note, that the CCM is only discovering the control-flow of a process; the (dynamic) discovery of the other perspectives and instance level information is described in later sections.

5.2.1 Preliminaries

Firstly, preliminary formal definitions are established which are the basis for formalising the methods and algorithms to solve the problem. For this the input and output formats, i.e. the BP event logs as well as the BP control-flow, are formalised. The example control-flow shown in Figure 2.12 on page 34 is used as a reference process in the remainder of this chapter. Formally, the control-flow of a business process is defined here as follows:

Definition 3 *A BP control-flow is a tuple* $CF = (A, S, J, \omega, \epsilon, C)$ *where* A *is a finite set of activities,* S *a finite set of splits,* J *a finite set of joins,* ω *the start event,* ϵ *the end event, and* $C \subseteq F \times F$ *the path connection relation, with* $F = A \cup S \cup J \cup \{\omega, \epsilon\}$*, such that*

- $C = \{(c_1, c_2) \in F \times F \mid c_1 \neq c_2 \wedge c_1 \neq \epsilon \wedge c_2 \neq \omega\}$,
- $\forall a \in A \cup J \cup \{\omega\} : |\{(a, b) \in C \mid b \in F\}| = 1$,
- $\forall a \in A \cup S \cup \{\epsilon\} : |\{(b, a) \in C \mid b \in F\}| = 1$,
- $\forall a \in J : |\{(b, a) \in C \mid b \in F\}| \geq 2$,
- $\forall a \in S : |\{(a, b) \in C \mid b \in F\}| \geq 2$, *and*
- *all elements* $e \in F$ *in the graph* (F, C) *are on a path from start event* ω *to end event* ϵ.

A block-structured control-flow as discussed in Section 2.4.1 is a refinement of Definition 3. It is additionally required that the process is hierarchically structured, i.e. every split element is mapped to exactly one join and vice-versa, both representing either a single entry or a single exit point of a non-sequence BP construct, e.g. Choice, Parallel, etc.

Furthermore, the input for the process discovery algorithm, i.e. a simple event log (see Section 2.4.1), is formally defined in the following:

Definition 4 *Let A be a finite set of activities then $\sigma \in A^*$ is a trace*[3] *and $Log \subseteq A^*$ is an event log, more specifically Log is a multi-set (bag)*[4] *of traces (sequences of activities). A finite sequence over A of length n is a mapping $\sigma : \{0,1,...,n-1\} \rightarrow A$ and is represented in the following by a string, i.e. $\sigma = [a_0, a_1, ..., a_{n-1}]$ where $a_i = \sigma(i)$ for $0 \leq i < n$. $|\sigma| = n$ denotes the length of the sequence.*

According to this definition $L_1 \in Log$ from Equation 2.1 on page 35 is an example of a simple event log of the reference BP and acts as the reference log in this chapter if not stated otherwise.

5.2.2 Divide and Conquer

The motivation to develop a discovery algorithm, which makes different BP constructs compete with each other for the best solution, is mainly derived from the challenge that logs often contain noise or even have frequent but conflicting behaviour. This cannot be expressed by common BP constructs without allowing duplicated activities. In uncertain cases the algorithm should look for the best solution which can support most of the behaviour, i.e. sequence, choice, parallelism, or loop or a combinations of these. Another important part of the CCM is that it is not based on local relationships like *direct neighbours* (see Section 2.4.2) but rather mines the process structure from global relationships between any two activities, e.g. which activities *eventually* follow one another in a trace. This approach has the benefit of avoiding a state-space explosion for logs of strongly connected BPs and will be of further benefit for the competition algorithm.

Algorithm 1 shows the conceptual methodology of the CCM algorithm in pseudocode. The CCM applies the divide-and-conquer paradigm and is implemented in a recursive fashion (see lines 7, 16, and 17). At the beginning `getFootprintAndBuildConstruct` is initially called for all involved activities ($A_m = A$) with the process bp consisting of only a start and end element. The recursive function is first creating a footprint fp from the given log L only considering the activities specified in set A_m (at the beginning all involved activities). In a next step it will be decided which is the best construct to represent the behaviour captured by fp: (1) If the activity set A_m only consists of one element, it will be decided which of the single activity constructs (see bottom of Figure 5.5) fits best - the process bp will then be enriched with the new single activity construct (see line 11); (2) If the activity set A_m contains more than one element, the suitability for each of the different constructs is calculated for any two activities $x, y \in A_m$ based on "soft" constraints and behaviour approximations, e.g. activities a and b are in a strong Sequence relationship.

[3] A^* is the set of finite sequences of elements of A.
[4] Since Log is a multi-set each trace can be contained more than once - see [233]

Algorithm 1: Methodology of the CCM in Pseudocode

```
   Data: Log L
   Result: CF bp
1  begin
2      A ← getSetOfAllActivitiesInLog(L);
3      CF bp ← buildInitialBPWithStartAndEnd();
4      bp ← getFootprintAndBuildConstruct(A, L, bp);
5      return bp;
6  Function getFootprintAndBuildConstruct(A_m, Log L, CF bp)
7      Footprint fp = extractFootprintForActivities(A_m, L);
8      if |A_m| = 1 then
9          Construct c ← analyseConstructForSingleActivity(fp);
10         bp ← createSingleActivityConstruct(c, A_m);
11     else
12         ConstructsSuitability[] cs ← calculateSuitabilityForConstructs(fp, A_m);
13         (Construct c, A_first, A_second) ← constructCompetition(cs, A_m);
14         bp ← createBlockConstruct(c, bp);
15         bp ← getFootprintAndBuildConstruct(A_first, L, bp);
16         bp ← getFootprintAndBuildConstruct(A_second, L, bp);
17     return bp;
```

The result of this calculation (line 13) is a number of suitability matrices, one for each construct. In the subsequent competition algorithm it is determined what is the best combination of (A) the construct type $c \in \{Sequence, Choice, Loop, ...\}$, and (B) the two subsets A_{first} and A_{second} of A_m with $A_{first} \cup A_{second} = A_m$, $A_{first} \cap A_{second} = \{\}$, and $A_{first}, A_{second} \neq \{\}$, that best accommodate all x, y-pair relations of the corresponding matrix of construct c (line 14). The construct is then created and added to the existing process model bp (line 15), e.g. XOR-split and -join if the winning construct c was *Choice*. At this stage the recursive method calls will be executed to analyse and construct the respective behaviour for the subsets A_{first} and A_{second}. The split up of the set A_m continues in a recursive fashion until it cannot be divided any more, i.e. the set consists of a single activity (see case (1)). The control-flow is completely constructed when the top recursive call returns.

The general principle of the algorithm is graphically depicted in Figure 5.2. Note, that the "Find Best Construct" component is equivalent to the function on line 6 of the Algorithm 1: `getFootprintAndBuildConstruct`. Furthermore, the first few steps to discover the example process from Figure 2.12 on page 34 are shown as coloured annotations (red, blue, and green text + boxes): (1) In the first step (red annotations) the best construct for all activities, i.e. $[a, b, c, d, e, f, g, h]$, is determined to be a *Decision* that splits the set of activities into two disjoint subsets, i.e. $[a, b, c, d, e]$ and $[f, g, h]$; (2) Then for both of these subsets their respective best suiting constructs are determined: *Sequence* for $[a, b, c, d, e]$ (green annotations) and *Loopover Sequence*[5] for $[f, g, h]$ (blue annotations). These con-

[5] This is a specialised BP construct that represents a sequence which can be repeatedly executed; All possible BP constructs are defined later in Section 5.2.4.

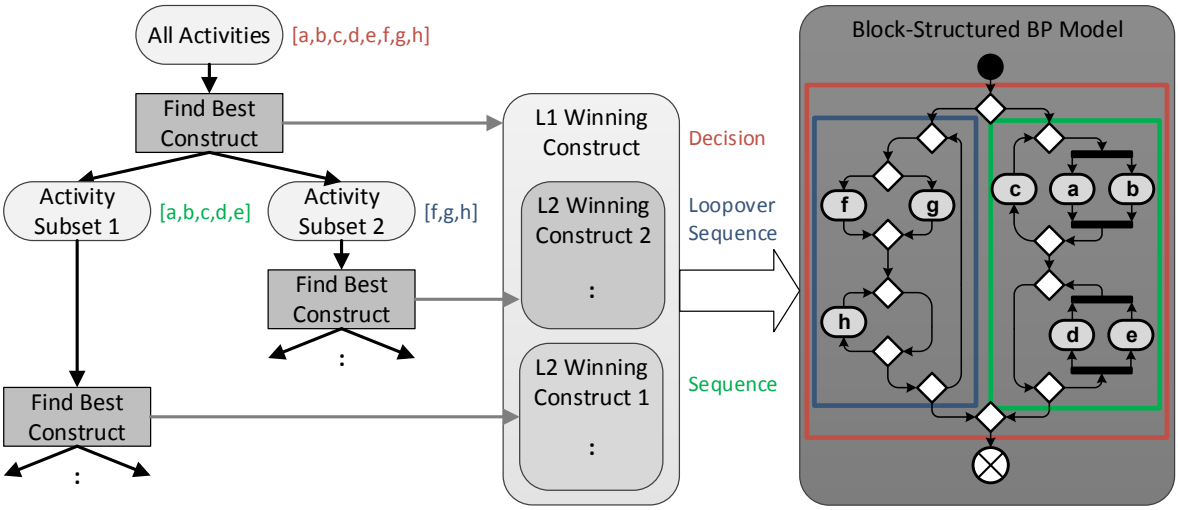


Fig. 5.2 First Steps of the Divide and Conquer Concept for the Example Process

structs further split up the respective sets of activities which are again analysed in a recursive fashion until the subsets consist of only one activity (not shown in the figure).

The functioning of "Find Best Construct" (or `getFootprintAndBuildConstruct`) can be divided into three different parts which are further explained following sections: (1) the calculation of the footprint (Section 5.2.3), (2) the suitability evaluation of the individual constructs (Section 5.2.4), and (3) the construct competition (Section 5.2.5).

5.2.3 Footprint

The CCM creates multiple footprints during its execution. These footprints are heuristic abstractions of the event log similar to the approach of the HeuristicsMiner (see Section 2.4.2) but based on different relations (global rather than local). At the beginning the overall footprint for all occurring activities has to be created. As the algorithm continues in its divide-and-conquer fashion, new activity subsets are built for each of which a new footprint has to be created, e.g. for $A = \{a,b,c,d,e\} : (a,b,c,d,e) \rightarrow ((a,b,c),(d,e)) \rightarrow (((a),(b,c)),((d),(e))) \rightarrow (((a),((b),(c))),((d),(e)))$[6] nine different footprints for sets $\{a,b,c,d,e\},\{a,b,c\},\{d,e\},\{b,c\},\{a\},\{b\},\{c\},\{d\},\{e\}$ need to be created. For reasons motivated earlier, the CCM focuses on global relations between the different activities (e.g. in how many traces will x be followed at some later point in the trace by activity y) and occurrence information about single activities (e.g. how many times does activity x appear in the log). The set of activities the footprint is to be calculated for is denoted by $A_m \subseteq A$. Furthermore, if the elements of A_m are encompassed by one or more parallelism constructs, two more sets need to be specified[7]:

- $A_i \subset A$ is the set of activities that are to be ignored, i.e. the occurrence of these activities do neither directly nor indirectly interfere with the occurrence of the activities

[6] (,) denote the nested blocks that emerge while splitting the sets recursively.

[7] For simplification reasons (in order to easier explain the overall concept) these two activity sets were not included in the pseudocode of Algorithm 1. Their functionality is instead explained in this section.

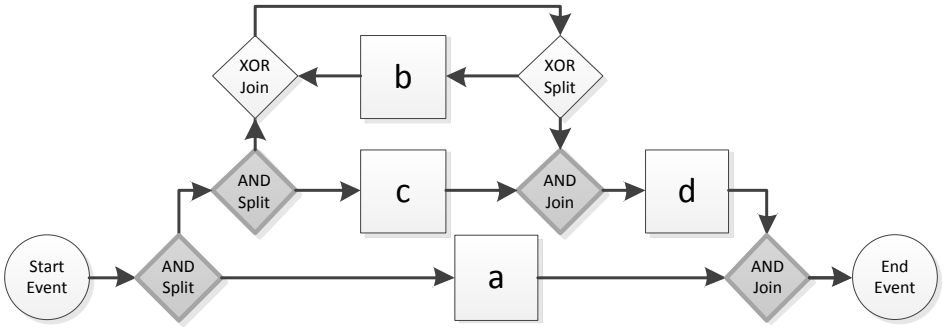


Fig. 5.3 Example business process with two nested parallel constructs

in A_m and are to be ignored (e.g. from a distant parallel path).

- $A_t \subset A$ is the set of activities that are to be tolerated, i.e. the occurrence of these activities do not directly interfere with the occurrence of the activities in A_m but indicate that the enclosing parallelism construct has been entered (e.g. from a local parallel path).

Note that A_m, A_i, A_t are disjoint sets of activities. A_i, A_t are empty if the activities in A_m are not on a parallel path. To distinguish between the different activity sets consider the process in Figure 5.3. For the creation of the footprint for the top path with activity b the three sets would be configured in the following way: $A_m = \{b\}, A_i = \{a\}, A_t = \{c\}$, because a is on a distant parallel path and c on a local parallel path in relation to b. The distinction between A_i and A_t is that elements in A_i are truly independent from the elements in A_m, but elements in A_t trigger the path in which elements of A_m reside as well. This is important to identify if a parallel path is optional like in the example where b does not appear in every trace.

To enable the CCM to discover the process from top till bottom the notion of a sub-trace is required which at a later stage will help to determine the footprint for a subset A_m of all the activities A included in the original log:

Definition 5 *Let $\sigma \in A^*$ be a trace, $A_m, A_i, A_t \subseteq A$ disjoint sets of activities, and $A_r = A \backslash (A_m \cup A_i \cup A_t)$ the set of activities in A but not in A_m, A_i, or A_t. Then $\lambda \in A_m^*$ is a sub-trace of σ ($\lambda \sqsubset^{A_m}_{A_i,A_t} \sigma$) iff there is $i, j \in \{0,1,...,|\sigma|-1\}$ and $i < j$ such that*

- $i = 0 \vee \sigma(i-1) \in A_r$ *and* $j = |\sigma|-1 \vee \sigma(j+1) \in A_r$ *and*
- $\exists_{l \in \{i,i+1,...,j\}} \sigma(l) \in (A_m \cup A_i)$ *and*
- $\forall_{l \in \{i,i+1,...,j\}} \sigma(l) \in (A_m \cup A_i \cup A_t)$ *and*
- $(|\lambda| = 0) \vee$
  $(|\lambda| = 1 \wedge \exists_{l \in \{i,i+1,...,j\}} (\sigma(l) = \lambda(0) \wedge \forall_{n \in \{i,i+1,...,j\}, n \neq l} \sigma(n) \notin A_m)) \vee$
  $(|\lambda| > 1 \wedge \forall_{k \in \{0,1,...,|\lambda|-2\}} \exists_{l,n \in \{i,i+1,...,j\}, l<n} (\sigma(l) = \lambda(k) \wedge \sigma(n) = \lambda(k+1) \wedge$
  $\forall_{p \in \{l+1,l+2,...,n-1\}} \sigma(p) \notin A_m))$.

Considering the example in Figure 5.3 and the corresponding trace $\sigma = [b,c,a,b,d]$ then $\lambda = [b,c,a,b,d]$ is a sub-trace of σ for $\sqsubset^{\{a,b,c,d\}}_{\{\},\{\}}$, $\lambda = [b,c,b,d]$ is a sub-trace of σ for $\sqsubset^{\{b,c,d\}}_{\{\},\{a\}}$, and $\lambda = [b,b]$ is a sub-trace of σ for $\sqsubset^{\{b\}}_{\{a\},\{c\}}$. In contrast, when instead considering trace $\sigma = [c,a,d]$ only the empty trace $\lambda = []$ represents the sub-sequence of σ for $\sqsubset^{\{b\}}_{\{a\},\{c\}}$ because a appeared and indicated that the top path has been enabled as well but exited without any occurrence of b. Note, that $\sigma \sqsubset^{A}_{\{\},\{\}} \sigma$, i.e. if the set of activities that are to be monitored is the set of all activities in the trace then the trace itself is the sub-trace. Also, in the case of a loop behaviour contained in a trace, the original trace may produce more than one sub-trace for a subset of activities that reside in the loop, e.g. for trace $\sigma = [b,a,c,a,b,c,b,a,d,e,e,d]$ from L_1 on page 35, the following three sequences are sub-traces of σ for $\sqsubset^{\{a,b\}}_{\{\},\{\}}$: $[b,a]$, $[a,b]$, and again $[b,a]$.

The purpose of the definition of a sub-trace is to establish the basis for the discovery of the best suited BP control-flow construct for complete traces but also for sub-traces corresponding to a subset of all involved activities. In order to build the footprint for sub-traces the following additional notations are introduced:

Definition 6 *Let $L \subseteq A^*$ be an event log over A, and $A_m, A_i, A_t \subseteq A$ disjoint sets of activities specifying the scope of the notations, and $\Lambda^{L,A_m}_{A_i,A_t} = \{\lambda \mid \lambda \in A^*_m \wedge \lambda \sqsubset^{A_m}_{A_i,A_t} \sigma \wedge \sigma \in L\}$ be a multi-set of all sub-traces in L specified by A_m, A_i, and A_t. Let activity $x \in A_m$, then is:*

1. $Once^{L,A_m}_{A_i,A_t}(x) = \{\lambda \in \Lambda^{L,A_m}_{A_i,A_t} \mid \exists_{i \in \{0,1,\ldots|\lambda|-1\}} \lambda(i) = x\}$,
2. $Sum^{L,A_m}_{A_i,A_t}(x) = \{(\lambda, l) \mid \lambda \in \Lambda^{L,A_m}_{A_i,A_t} \wedge \lambda(l) = x\}$,
3. $Start^{L,A_m}_{A_i,A_t}(x) = \{\lambda \in \Lambda^{L,A_m}_{A_i,A_t} \mid \lambda(0) = x\}$,
4. $|Once^{L,A_m}_{A_i,A_t}(x)|$, $|Sum^{L,A_m}_{A_i,A_t}(x)|$, *and* $|Start^{L,A_m}_{A_i,A_t}(x)|$ *the sizes of these sets.*

Furthermore, let $x, y \in A_m$, then is:

1. $x >^{L,A_m}_{A_i,A_t} y$ *iff a sub-trace* $\lambda \in \Lambda^{L,A_m}_{A_i,A_t}$ *and* $i, j \in \{0,1,\ldots,|\lambda|-1\}$ *and* $i < j$ *exists such that* $\lambda(i) = x$ *and* $\lambda(j) = y$ *and* $\forall_{k \in \{0,1,\ldots,j-1\}} \lambda(k) \neq y$,
2. $x >>^{L,A_m}_{A_i,A_t} y$ *iff a sub-trace* $\lambda \in \Lambda^{L,A_m}_{A_i,A_t}$ *and* $i, j \in \{0,1,\ldots,|\lambda|-1\}$ *and* $i < j$ *exists such that* $\lambda(i) = x$ *and* $\lambda(j) = y$,
3. $|x >^{L,A_m}_{A_i,A_t} y|$ *the number of occurrences of* $x >^{L,A_m}_{A_i,A_t} y$ *in* L,
4. $|x >>^{L,A_m}_{A_i,A_t} y|$ *the number of occurrences of* $x >>^{L,A_m}_{A_i,A_t} y$ *in* L.

Example: For $A_m = A = \{a,b,c,d\}$ consider the following log $L_3 = \{[a,b,c,d]^2, [b,a,c,b,d]^1\}$:

- $|Once^{L,A_m}_{A_i,A_t}(x)|$ determines how many of the sub-traces contained x,
  e.g. $|Once^{L_3,\{a,b,c,d\}}_{\{\},\{\}}(b)| = 3$ (twice from $[a,b,c,d]^2$ and once from $[b,a,c,b,d]^1$);
- $|Sum^{L,A_m}_{A_i,A_t}(x)|$ represents how many x were in all sub-traces,
  e.g. $|Sum^{L_3,\{a,b,c,d\}}_{\{\},\{\}}(b)| = 4$ (2 from $[a,b,c,d]^2$ + 2 from $[b,a,c,b,d]^1$);

- $|Start^{L,A_m}_{A_i,A_t}(x)|$ specifies how many times the sub-trace started with x,
  e.g. $|Start^{L_3,\{a,b,c,d\}}_{\{\},\{\}}(b)| = 1$ (only $[b,a,c,b,d]^1$ started with b)
- $|x >^{L,A_m}_{A_i,A_t} y|$ determines the amount of sub-traces in which x at some point appeared before the first occurrence of y,
  e.g. $|a >^{L_3,\{a,b,c,d\}}_{\{\},\{\}} b| = 2$ (only in $[a,b,c,d]^2$ a appears before the first b)
- $|x >>^{L,A_m}_{A_i,A_t} y|$ determines the amount of sub-traces in which x is occurring at some point before any y,
  e.g. $|a >>^{L_3,\{a,b,c,d\}}_{\{\},\{\}} b| = 3$ (twice from $[a,b,c,d]^2$ and once from $[b,a,c,b,d]^1$).

With the help of these absolute values the footprint can now be calculated by putting them in relation to the number of all sub-traces. Then based on these values the CCM performs a construct analysis which in turn enables the execution of the competition between these constructs.

Definition 7 *Let $L \subseteq A^*$ be an event log over A, $A_m, A_i, A_t \subseteq A$ disjoint sets of activities specifying the scope of the footprint, $|\Lambda^{L,A_m}_{A_i,A_t}|$ be the number of sub-traces in L specified by A_m, A_i, and A_t. Let $x \in A_m$:*

- *The **occurrence once** value $Oon^{L,A_m}_{A_i,A_t}(x)$ and the **occurrence overall** value $Oov^{L,A_m}_{A_i,A_t}(x)$ are calculated as follows:*

$$Oon^{L,A_m}_{A_i,A_t}(x) = \frac{|Once^{L,A_m}_{A_i,A_t}(x)|}{|\Lambda^{L,A_m}_{A_i,A_t}|} \qquad Oov^{L,A_m}_{A_i,A_t}(x) = \frac{|Sum^{L,A_m}_{A_i,A_t}(x)|}{|\Lambda^{L,A_m}_{A_i,A_t}|} \tag{5.3}$$

- *The **first element** value $Fel^{L,A_m}_{A_i,A_t}(x)$ is calculated with the following equation:*

$$Fel^{L,A_m}_{A_i,A_t}(x) = \frac{|Start^{L,A_m}_{A_i,A_t}(x)|}{|\Lambda^{L,A_m}_{A_i,A_t}|} \tag{5.4}$$

*Let $x, y \in A_m$ then is the **appears before first** value $x \rhd^{L,A_m}_{A_i,A_t} y$ and the **appears before** value $x \rhd\rhd^{L,A_m}_{A_i,A_t} y$ calculated as follows:*

$$x \rhd^{L,A_m}_{A_i,A_t} y = \frac{|x >^{L,A_m}_{A_i,A_t} y|}{|\Lambda^{L,A_m}_{A_i,A_t}|} \qquad x \rhd\rhd^{L,A_m}_{A_i,A_t} y = \frac{|x >>^{L,A_m}_{A_i,A_t} y|}{|\Lambda^{L,A_m}_{A_i,A_t}|} \tag{5.5}$$

All values of $Oon^{L,A_m}_{A_i,A_t}$, $Fel^{L,A_m}_{A_i,A_t}$, $\rhd^{L,A_m}_{A_i,A_t}$, and $\rhd\rhd^{L,A_m}_{A_i,A_t}$ will be ≥ 0 and ≤ 1 since each of their relation can occur at most once per sub-trace. However, the values of $Oov^{L,A_m}_{A_i,A_t}(x)$ can become greater than 1 if activity x occurs on average more than once per sub-trace. The complete footprint consisting of *Oon*, *Oov*, *Fel*, $\rhd$, and $\rhd\rhd$ is displayed in this thesis as labelled vectors for the values of *Oon*, *Oov*, and *Fel* and as labelled matrices for the values

of $\triangleright$ and $\triangleright\triangleright$. Considering again the example log L_1 from page 35, its complete footprint $FP^{L_1,A}_{\{\},\{\}}$ for $A_m = A = \{a,b,c,d,e,f,g,h\}$ is:

$$Oon^{L_1,A}_{\{\},\{\}}(x): \begin{array}{cccccccc} a & b & c & d & e & f & g & h \\ \end{array} \begin{pmatrix} 0.56 & 0.56 & 0.18 & 0.44 & 0.44 & 0.38 & 0.35 & 0.44 \end{pmatrix}$$

$$Oov^{L_1,A}_{\{\},\{\}}(x): \begin{array}{cccccccc} a & b & c & d & e & f & g & h \\ \end{array} \begin{pmatrix} 0.91 & 0.91 & 0.35 & 0.62 & 0.62 & 1.09 & 0.65 & 0.85 \end{pmatrix}$$

$$Fel^{L_1,A}_{\{\},\{\}}(x): \begin{array}{cccccccc} a & b & c & d & e & f & g & h \\ \end{array} \begin{pmatrix} 0.15 & 0.41 & 0.00 & 0.00 & 0.00 & 0.32 & 0.12 & 0.00 \end{pmatrix}$$

$x \triangleright\triangleright^{L_1,A}_{\{\},\{\}} y$:

	a	b	c	d	e	f	g	h
a	0.18	0.32	0.18	0.44	0.44	0	0	0
b	0.41	0.18	0.18	0.44	0.44	0	0	0
c	0.18	0.18	0.18	0.18	0.18	0	0	0
d	0	0	0	0.18	0.32	0	0	0
e	0	0	0	0.29	0.18	0	0	0
f	0	0	0	0	0	0.24	0.24	0.32
g	0	0	0	0	0	0.29	0.29	0.29
h	0	0	0	0	0	0.29	0.24	0.24

$x \triangleright^{L_1,A}_{\{\},\{\}} y$:

	a	b	c	d	e	f	g	h
a	0	0.15	0.18	0.44	0.44	0	0	0
b	0.41	0	0.18	0.44	0.44	0	0	0
c	0	0	0	0.18	0.18	0	0	0
d	0	0	0	0	0.32	0	0	0
e	0	0	0	0.18	0	0	0	0
f	0	0	0	0	0	0	0.24	0.32
g	0	0	0	0	0	0.06	0	0.06
h	0	0	0	0	0	0.06	0.24	0

Definition 8 *Let $L \subseteq A^*$ be an event log over A, $A_m, A_i, A_t \subseteq A$ disjoint sets of activities specifying the scope of the footprint FP, then is*

$$FP^{L,A_m}_{A_i,A_t} = (Oon^{L,A_m}_{A_i,A_t}, Oov^{L,A_m}_{A_i,A_t}, Fel^{L,A_m}_{A_i,A_t}, \triangleright^{L,A_m}_{A_i,A_t}, \triangleright\triangleright^{L,A_m}_{A_i,A_t}) \tag{5.6}$$

Note: Since the analysis in the remainder of this section is based on one specific footprint $FP^{L,A_m}_{A_i,A_t}$, if not otherwise stated $FP^{L,A_m}_{A_i,A_t}$, $Oon^{L,A_m}_{A_i,A_t}$, $Oov^{L,A_m}_{A_i,A_t}$, $Fel^{L,A_m}_{A_i,A_t}$, $\triangleright^{L,A_m}_{A_i,A_t}$, and $\triangleright\triangleright^{L,A_m}_{A_i,A_t}$ will be simply denoted as FP, Oon, Oov, Fel, $\triangleright$, and $\triangleright\triangleright$ for the remainder of this section to support the readability, i.e. with regards to Equation 5.6: $FP = (Oon, Oov, Fel, \triangleright, \triangleright\triangleright)$.

5.2.4 Suitability of BP-Constructs

The next step is to infer to what degree the (sub-)footprint FP is reflected by each BP construct, respectively. If the above footprint is considered it can already logically inferred that the activity sets $\{a,b,c,d,e\}$ and $\{f,g,h\}$ are in a Choice construct because all values between the two sets in the $\triangleright\triangleright$ matrix are 0[8]. It can be additionally inferred that in the $\triangleright\triangleright$ matrix $\{a,b,c\}$ and $\{d,e\}$ are in a Sequence construct because d and e are never followed by a, b, or c, but a, b, and c can be followed by d or e. Note, that the footprint FP is a "perfect" example to showcase how BP constructs can be derived from such a footprint. Usually, footprint matrices are not sorted in a way that this can be manually inferred by

[8] i.e. none the of activities $\{f,g,h\}$ ever follows or is followed by any of the activities $\{a,b,c,d,e\}$ in a trace

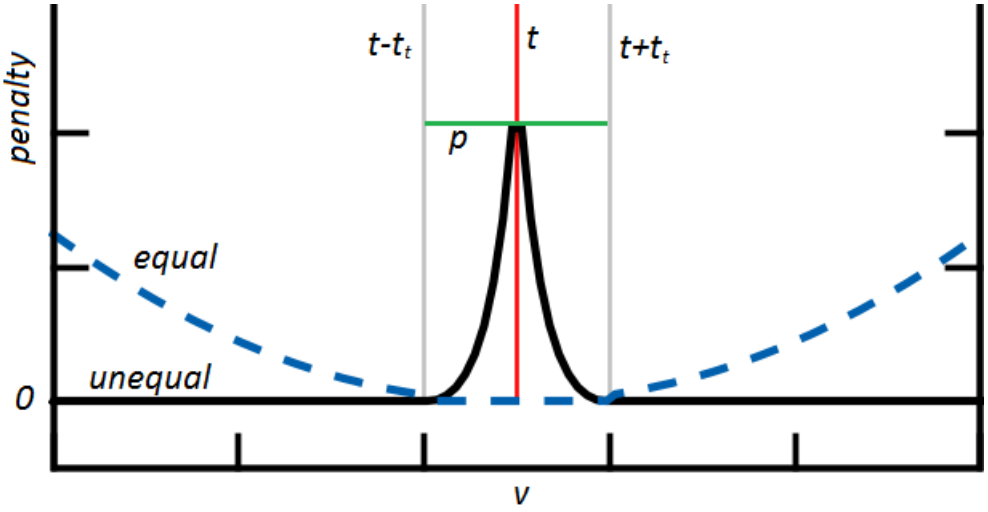


Fig. 5.4 Penalty development of ($v \cong t$) (equal) and ($v \ncong t$) (unequal) for given p_u and t_t

logical deduction and noise as well as the incompleteness of the original log make it more difficult to find the right construct since none would fit perfectly unlike in the example.

The CCM works similarly to how the Decision and Sequence constructs were identified for the example footprint: Based on the footprint *FP*, the algorithm identifies the BP construct that best describes the footprint with the help of constraints. The construct which fulfils its respective constraints best is chosen to be part of the BP control-flow model. As an example, the constraint for the Choice construct (i.e. two activities being mutually exclusive) requires the appears-before values between the respective activities to be "equal to 0". However, in order to handle noise well, the CCM uses the following definition of equality for checking the fulfilment of constraints:

Definition 9 *Let v be the actual value, t be the target value, p_u be the maximum penalty for a not fulfilled unequal relation, t_t the tolerance which determines the maximum difference so that v and t are still considered equal, and $v, t, p_u, t_t \in \mathbb{R}^+$:*

$$
\begin{aligned}
(v \ncong t) &= \begin{cases} p_u * (\frac{2*t_t}{|v-t|+t_t} - 1)^2 & if \quad |v-t| < t_t \\ 0 & else \end{cases} \\
(v \cong t) &= \begin{cases} 0 & if \quad |v-t| < t_t \\ |v-t|^2 & else \end{cases}
\end{aligned}
\tag{5.7}
$$

Figure 5.4 shows the development of the functions ($v \cong t$) (stroked blue line) and ($v \ncong t$) (black line) for a given p_u and t_t. The (penalty) value of ($v \cong t$) rises the further the actual value v is away from the target value t. In contrast, the (penalty) value of ($v \ncong t$) rises (up to p_u) the closer the actual value v is to the target value t. This essentially translates "hard" constraints, e.g. the constraint is satisfied if v equals 0, into "soft" constraints, e.g. the constraints satisfaction penalty value is 0.3. Note, that a penalty of 0 indicates that the constraint is assumed to be fully satisfied whereas a high the penalty value indicates

that the constraint is only little satisfied.

Construct Suitability for a Single Activity: If a footprint consists of only one activity, i.e. $|A_m| = 1$, no competition between constructs is necessary. Instead, the correct construct is identified based on the values in the footprint. Four different constructs for a single activity $x \in A_m$ exist[9]:

- *Normal*: if $(Fel(x) \cong 1) = 0$ and $(x \rhd \rhd x \cong 0) = 0$ then x is a simple activity.
- *Optional*: if $(Fel(x) \cong 1) > 0$ and $(x \rhd \rhd x \cong 0) = 0$ then x is an optional activity, i.e. x may also be skipped.
- *Loopover*: if $(Fel(x) \cong 1) = 0$ and $(x \rhd \rhd x \cong 0) > 0$ then x is a looping activity, i.e. x can repeatedly occur after itself ("short loop").
- *Loopback*: if $(Fel(x) \cong 1) > 0$ and $(x \rhd \rhd x \cong 0) > 0$ then x is an optional looping activity, i.e. x may be skipped but can also repeatedly occur.

Construct Suitability for Multiple Activities: If a footprint *FP* consists of more than one activity, i.e. $|A_m| > 1$, a preliminary analysis is carried out to identify the suitability of any two activities $x, y \in A_m$ with regards to each available construct, e.g. activities x and y are in a very strong Parallelism relation but less strong in a Sequence relation. The calculation of this construct's suitability is again based on constraints. If a constraint is not fulfilled there will be a penalty depending on the "level" of the constraint[10] and how badly it has failed.

The first step of the suitability analysis is to identify if the construct represented by the *FP* is optional, i.e. an optional path exists that allows to bypass this construct. If this is the case the *FP* needs to be normalised, i.e. removal of the overall optional behaviour. For this purpose it is calculated if and to what extent the *FP* also recorded empty (sub-)traces, i.e. relative occurrence of an empty (sub-)trace: $op = 1 - \sum_{x \in A_m} Fel(x)$. The influence of these empty traces is removed from the *FP* by multiplying every value of $Oon, Oov, Fel, \rhd$, and $\rhd\rhd$ with $\frac{1}{1-op}$.

Additionally, the following values are calculated for each $x \in A_m$ and each $x, y \in A_m$ pair:

Definition 10 *Let $x \in A_m$ be an activity recorded in FP then is the repetition of x in FP denoted $rep(x) = \frac{Oov(x) - Oon(x)}{Oov(x)}$.*

Definition 11 *Let $x, y \in A_m$ be activities recorded in FP then is:*
(1) the ratio of (sub-)traces in which both activities x and y appear: $oc(x, y) = x \rhd y + y \rhd x$,
(2) the maximum possible probability of x and y appearing in the same (sub-)trace:

[9] $>$ is in this case the common "greater than" relation, not the one specified in Definition 6
[10] the levels of constraints will be discussed later in this section

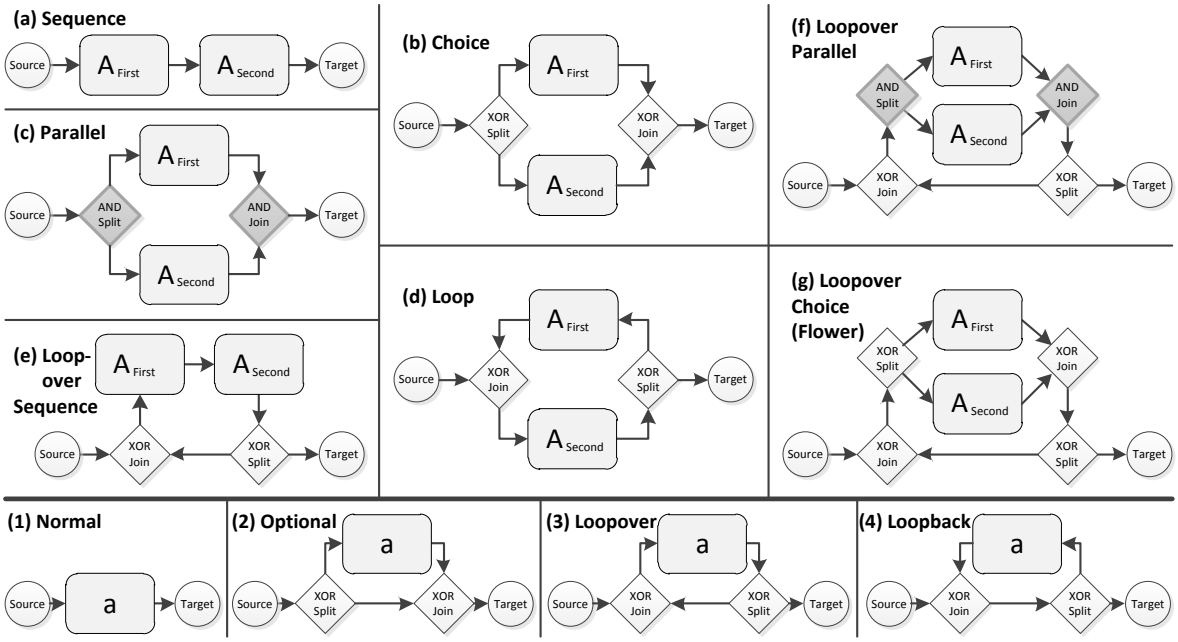


Fig. 5.5 Supported Business Process Constructs

$moc(x,y) = min(Oon(x), Oon(y))$, *and*
(3) the combined occurrence probability of x *and* y*:* $coc(x,y) = Oon(x) * Oon(y)$.

The algorithm supports the identification of the BP constructs shown in Figure 5.5. For each construct a set of constraints have been formulated to determine to which degree a construct represents the global relation between any two activities. In Table 5.1 the constraints for each construct are listed, sorted by the constraint level. It is distinguished between different levels/severities of constraints to highlight the importance of their fulfilment:

- **Strict**: Constraints of this type can be seen as "iff" requirements on the construct and are thus required to be fulfilled for a construct to apply, e.g. every activity in a Loop construct has to occur at least once repeatedly in a trace, otherwise the observed construct cannot be a Loop. Penalties originating from the failure of constraints of this type have a strong influence on the suitability of a construct relation;

- **Log-Complete**: A log-complete constraint is fulfilled when all variants are represented in the log, i.e. the log is complete. If the log is incomplete constraints of this type may fail. This is why penalties originating from the failure of log-complete constraints have a medium influence on the suitability of a construct relation;

- **Indication**: An indication constraint represents default behaviour of the construct but may not be fulfilled even in a complete log. Indication constraints are basically not constraints in the common sense but rather approximations of default behaviour in order to distinguish between two very similar constructs, e.g. Parallel and Loopover-Parallel. Penalties originating from the failure of constraints of this type have a low influence on the overall suitability.

Table 5.1 Supported BP constructs and their constraints sorted by constraint level

Construct	Strict	Log-Complete	Indication
Choice	$x \rhd \rhd y \cong 0$, $y \rhd \rhd x \cong 0$	-	-
Sequence	$x \rhd \rhd y \ncong 0$, $y \rhd \rhd x \cong 0$, $x \rhd \rhd y \cong x \rhd y$	-	-
Parallel	-	$x \rhd y \ncong 0$, $y \rhd x \ncong 0$, $coc(x,y) \cong oc(x,y)$	$x \rhd \rhd y \cong (x \rhd y + min(rep(x), rep(y))) * (moc(x,y) - x \rhd y)$
Loop	$rep(x) \ncong 0$, $rep(y) \ncong 0$	$x \rhd \rhd y \cong coc(x,y)$, $y \rhd \rhd x \cong coc(x,y)$	-
Loopover-Sequence	$rep(x) \ncong 0$, $rep(y) \ncong 0$	$x \rhd \rhd y \cong coc(x,y)$, $y \rhd \rhd x \ncong coc(x,y)$, $x \rhd \rhd y \ncong y \rhd \rhd x$	-
Loopover-Parallel	$rep(x) \ncong 0$, $rep(y) \ncong 0$	$x \rhd y \ncong 0$, $y \rhd x \ncong 0$, $coc(x,y) \cong oc(x,y)$	$x \rhd \rhd y \cong (x \rhd y * Oon(y) + Oov(y) - Oon(y)) / Oov(y)$
Loopover-Choice (Flower)	$rep(x) \ncong 0$, $rep(y) \ncong 0$, $coc(x,y) \ncong 1$	$x \rhd y \ncong 0$, $y \rhd x \ncong 0$	$x \rhd \rhd y \cong y \rhd \rhd x$, $coc(x,y) \cong max(0, Oon(y) + Oon(y) - 1)$

Based on the constraints listed in Table 5.1 and their respective constraint level the suitability of each construct for any activity pair $x, y \in A_m, x \neq y$ is calculated. Exemplary, it is shown in the following how values for the constructs Choice $Ch(x,y)$ and Sequence $Se(x,y)$ are calculated ($w_s \in \mathbb{R}$ is the weight of Strict constraints):

1. $Ch(x,y) = w_s * \frac{1}{2} * (x \rhd \rhd y \cong 0 + y \rhd \rhd x \cong 0)$,
2. $Se(x,y) = w_s * \frac{1}{3} * (x \rhd \rhd y \ncong 0 + y \rhd \rhd x \cong 0 + x \rhd \rhd y \cong x \rhd y)$.

The values for the other constructs are similarly calculated, with w_{lc}, $w_i \in \mathbb{R}$ further specifying the weights for Log-Complete and Indication constraints, respectively. For the footprint $FP^{L_1,A}_{\{\},\{\}}$ on page 144 the resulting suitability matrices Ch and Se for $w_s = 0.6$ are:

$Ch(x,y)$:

	a	b	c	d	e	f	g	h
a	–	0.26	0.6	0.3	0.3	0	0	0
b	0.26	–	0.6	0.3	0.3	0	0	0
c	0.6	0.6	–	0.3	0.3	0	0	0
d	0.3	0.3	0.3	–	0.29	0	0	0
e	0.3	0.3	0.3	0.29	–	0	0	0
f	0	0	0	0	0	–	0.34	0.39
g	0	0	0	0	0	0.34	–	0.34
h	0	0	0	0	0	0.39	0.34	–

$Se(x,y)$:

	a	b	c	d	e	f	g	h
a	–	0.13	0.2	0	0	0.2	0.2	0.2
b	0.07	–	0.2	0	0	0.2	0.2	0.2
c	0.4	0.4	–	0	0	0.2	0.2	0.2
d	0.4	0.4	0.4	–	0.09	0.2	0.2	0.2
e	0.4	0.4	0.4	0.14	–	0.2	0.2	0.2
f	0.2	0.2	0.2	0.2	0.2	–	0.14	0.12
g	0.2	0.2	0.2	0.2	0.2	0.18	–	0.18
h	0.2	0.2	0.2	0.2	0.2	0.22	0.22	–

Algorithm 2: Competition Algorithm for the Choice Construct

Data: A, Ch
Result: A_{first}, A_{second}

```
1  begin
2    PriorityQueue openCases ← {};
3    openCases.add(({}, {}, A, .0));
4    while true do
5      (A_first, A_second, A_left, pen) ← openCases.poll();
6      if A_left = {} then return(A_first, A_second) ;
7      x ← A_left.poll();
8      if |A_left| > 0 ∨ |A_second| > 0 then
9        A_new ← A_first ∪ {x};
10       pen_new ← 0;
11       foreach y ∈ A_second do pen_new ← pen_new + Ch(x, y) ;
12       if pen_new > 0 then pen_new ← pen_new/|A_second| + pen ;
13       else pen_new ← pen ;
14       openCases.add((A_new, A_second, A_left, pen_new));
15     end
16     if |A_left| > 0 ∨ |A_first| > 0 then
17       A_new ← A_second ∪ {x};
18       pen_new ← 0;
19       foreach y ∈ A_first do pen_new ← pen_new + Ch(y, x) ;
20       if pen_new > 0 then pen_new ← pen_new/|A_first| + pen ;
21       else pen_new ← pen ;
22       openCases.add((A_first, A_new, A_left, pen_new));
23     end
24   end
25 end
```

5.2.5 Competition Algorithm

The goal of the competition algorithm is to find the best combination of (1) the construct type, e.g. Sequence, Choice, or Loop, and (2) the best two subsets A_{first} and A_{second} of A with $A_{first} \cup A_{second} = A$ and $A_{first} \cap A_{second} = \{\}$, that best accommodates all corresponding x, y-pair relations. The principal of operation of the competition algorithm is shown by means of the Choice construct, i.e. for the explanation the construct is fixed and the two subsets A_{first} and A_{second} have to be determined. A naive solution would be to create and compare all possible split ups. With regards to the execution time of the CCM, this is not desirable since the algorithm would have to check all $2^{|A|} - 1$ possible split ups. Instead it should be taken advantage of the fact that these relations, in this case only Ch, represent the global relation of x and y. That means it is irrelevant for the calculation of the penalty what the relations between the elements in the same set are (either A_{first} and A_{second}). In Algorithm 2 it is presented how the competition algorithm works if only Ch is considered to be part of the competition. Note, that the priority queue is ordered firstly by the penalty value pen and secondly by how even the split up is (it is desirable to split the activity set as evenly as possible to quickly reduce the number of activities). For Ch from the exam-

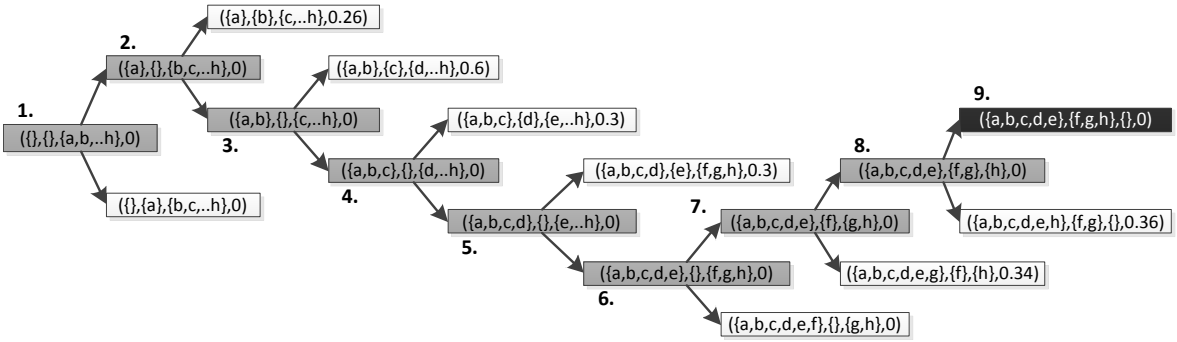


Fig. 5.6 Competition Algorithm: Traversing to the best split up

ple log with $A_m = A = \{a,b,c,d,e,f,g,h\}$ the algorithm functions as follows: in first step an "empty" combination tuple (A_{first}, A_{second}, A_{left}, pen) is inserted into the priority queue with (1) A_{first} and A_{second}, the both disjunct sets of activities are empty at the beginning; (2) the set of the activities A_{left} which contains the activities that still have to be assigned to either the first or second set - $A_{left} = A = \{a,b,c,d,e,f,g,h\}$; (3) the current penalty $pen = 0$. With this one element in the priority queue the while-loop is entered. There, the tuple with the highest priority (the one that was just inserted) is removed from the queue and further processed. That means in this case that an activity is removed from A_{left} and assigned to x. Now, two more tuples are created, one with x in A_{first} (line 8-15) and one with x in A_{second} (line 16-23). The conditions on lines 8 and 16 are to ensure that at the last assignment (when A_{left} is empty) it will only be added to either (A_{first} or A_{second} when the other set is not empty. This ensures that only real split ups are allowed where neither subset is empty. According to the set x was inserted into, all Ch values from x to elements from the other set are checked and the average of these is added to the respective penalty value pen. Both newly created tuples are then inserted into the priority queue. This continues until the best combination tuple has no activities left, i.e. $A_{left} = \{\}$. In Figure 5.6 the different created combinations for the choice competition are shown: the light grey combinations are still in the queue when the algorithm terminates, the grey combinations are already processed and the number next to them represents the order in which they were processed; the black combination is the winner of the competition algorithm.

Note, that the competition has only been carried out for the choice construct in order to show how the splitting up is achieved. More BP constructs can enter the competition by three simple modifications of the algorithm: (1) the tuple in the priority queue also has to contain the construct type, e.g. Choice, Loopover-Sequence,etc. (2) adding one "empty" tuple per construct to the priority queue before the while loop is entered; (3) The penalty calculation then has to be carried out on the relation matrix corresponding to the currently processed construct type. Note, that in the default configuration all BP constructs enter the competition.

When the winning construct and the corresponding split up (for sets with multiple activities) has been decided the construct is created and incorporated into the BP control-flow model (see Algorithm 1 on page 139). For this the previously identified optional

empty paths (their influence was removed from the suitability matrices - see Section 5.2.4, page 146) are taken into account. Also, if possible, the identified BP construct is combined with the parent construct, e.g. if the parent construct was a choice and the now identified construct is a choice again, then no new decision and merge elements are created but instead the parent decision construct is extended.

In conclusion, in this section the Construct Competition Miner (CCM) was introduced. The CCM is a deterministic process discovery algorithm that operates in a static fashion and follows a divide-and-conquer approach which, from a given event log, directly mines a block-structured BP control-flow model that represents the main behaviour of the process. The CCM has the following main features: (1) A deadlock-free, block-structured business process without duplicated activities is mined; (2) The following BP constructs are supported and can be discovered for single activities: Normal, Optional, Loopover, and Loopback; or for a set of activities: Choice, Sequence, Parallel, Loop, Loopover-Sequence, Loopover-Choice, Loopover-Parallel (see Figure 5.5), and additionally all of them as optional constructs - these are constructs supported by the majority of business process standards like BPMN or YAWL; (3) The CCM operates on "global" relationships between activities rather than "local" relationships to determine the suitability of the respective BP constructs; (4) If conflicting, incomplete, or exceptional behaviour exists in the log, the CCM picks the "best" fitting BP construct.

5.3 Dynamic Process Discovery Framework

The CCM is specialised on the discovery of the control-flow perspective from an event log in a static way. This is not sufficient to meet all requirements for establishing a causal connection between BPMS and DBPMRT in a dynamic run-time environment (see Section 5.1), e.g. because there is no concept for detecting change or applicability on an event stream. The basic problem to overcome in order to meet these requirements is that of abstraction. This is especially the case for the control-flow of a BP for which elaborate algorithms like the CCM and other state-of-the-art approaches (see Section 2.4.2) exist but rely on time-intensive aggregations. This is why a new approach towards a process discovery based on event-based processing needs to be established, e.g. by adapting existing algorithms (like the CCM) to enable direct processing of an event stream (a number of initial approaches in this area have been discussed in Section 2.4.4). Furthermore, process discovery approaches need to be extended to allow for the discovery of all perspectives (not just the control-flow) and instance information in order to accurately and holistically reflect the system and allow for advanced reasoning (see Chapter 6). According to the defined requirements such a monitoring system should be also extensible to allow for different purposes and individual monitoring specialised on specific areas of interest.

For this reason the Dynamic Process Discovery Framework (DPDF) has been developed. The DPDF can be characterised as a modular framework for monitoring one or

more BPMSs in order to provide at any point in time a reasonably accurate representation of the current state of the processes deployed in the systems with regards to their control-flow, resource, and performance perspectives as well as the state of still open traces. That means, change in the mentioned aspects of processes in the system during run-time has to be reflected in the monitored representation of the current state, i.e. the DBPMRT. In this section the general concept of the DPDF along with its components and model artefacts are described. The individual details of the components are further specified in the sections thereafter.

5.3.1 Concept

The main concept of the DPDF is to introduce an intermediate layer of abstraction, i.e. the *dynamic footprint*, and to consequently distinguish two different lifecycles and their associated components:

1. The *Event Processing* operating at run-time, complying to the stringent requirements with regards to the algorithmic run-time, and producing a so called *dynamic footprint* from the event input.

2. The *Footprint Interpretation* which can compute the actual state of the business process based on the current dynamic footprint. The footprint interpretation has less restrictions with regards to algorithmic run-time as it does not have to be executed with every occurring event but rather more autonomously, i.e. either on demand, or repeatedly after a certain time has passed or after a fixed number of events or traces occurred.

The concept of the resulting framework is presented in an information flow diagram in Figure 5.7. It shows agents in a rectangular shape and models with round edges. The general concept works as follows[11]: Events from different sources of the monitored *Enterprise System*, in which the end-to-end process is deployed, are processed to a standardised format and put into a global context by the *Event Hub*. The standardised events are then further processed to update the current dynamic footprint, which acts as the current state of the process. The footprint information can then at any point in time be compiled to the actual state information (instance and type level) of the business process, i.e. the abstract footprint representation is interpreted into knowledge conforming to a generalised business process standard (control-flow, performance, and resource perspective, and state of the active traces). This information can be processed by different reasoning algorithms to further analyse the process, e.g. performance prediction via simulation (see Chapter 6).

In the following the models and agents involved in the DPDF framework are described in more detail. This includes descriptions further specifying the involved components

[11] To improve the understanding for the reader the explanation of the concept focuses on the monitoring of one end-to-end process only.

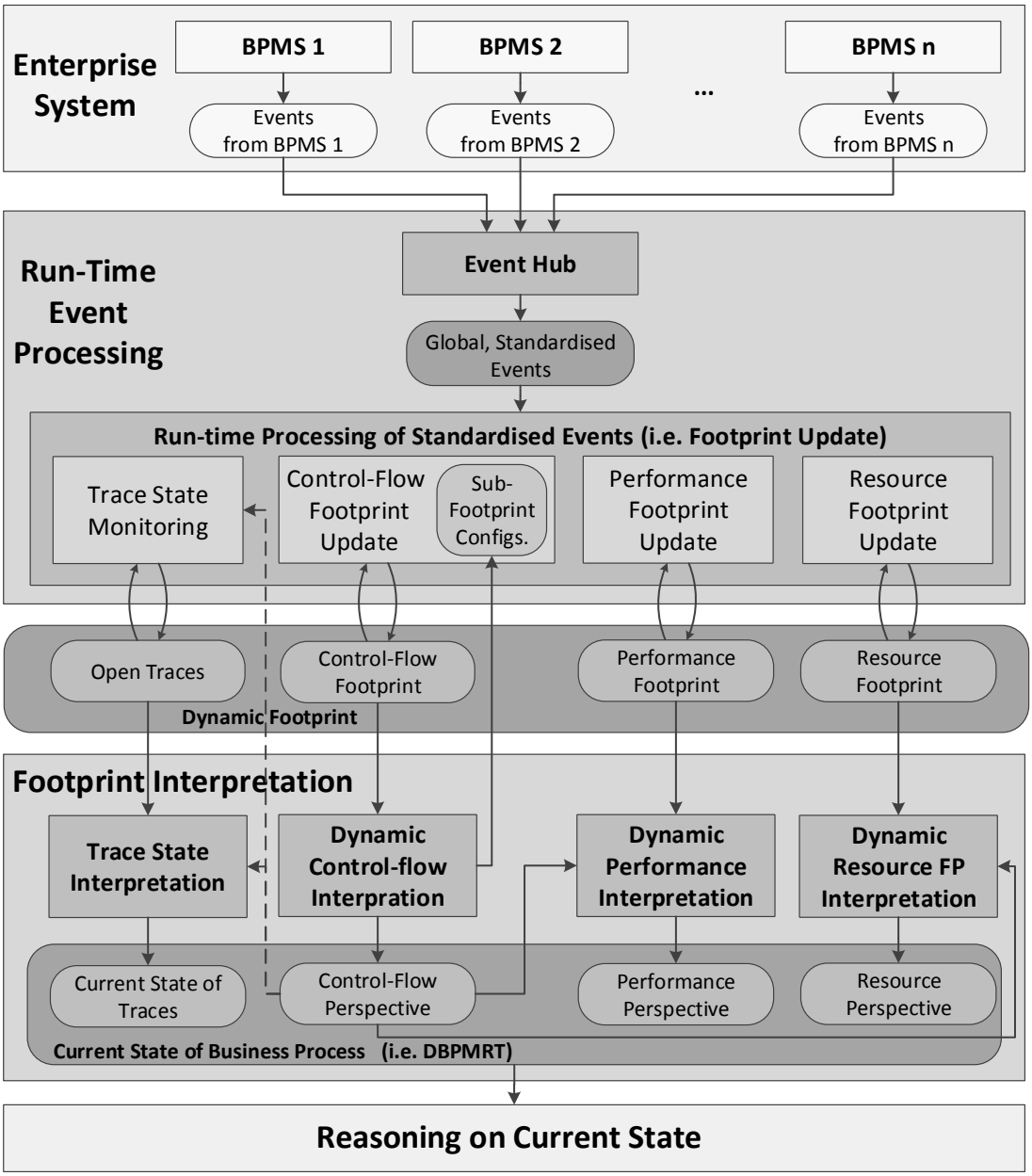


Fig. 5.7 Information Flow: Agents and Model Artefacts involved in the DPDF Framework

and models as well as remarks on important implementation details for artefacts that are not explicitly explained at a later stage.

5.3.2 Event Hub and Global, Standardised Events

The *Event Hub* and the *Global, Standardised Events* are part of the DPDF mainly with the purpose of complying to the Extensibility requirement TR6. They are responsible for the pre-processing step in which every event is translated into a standardised version that can be further processed.

Global, Standardised Events In Section 2.2.2 the common elements and the lifecycle of BP event logs are introduced. The *Global, Standardised Events* are pre-processed events

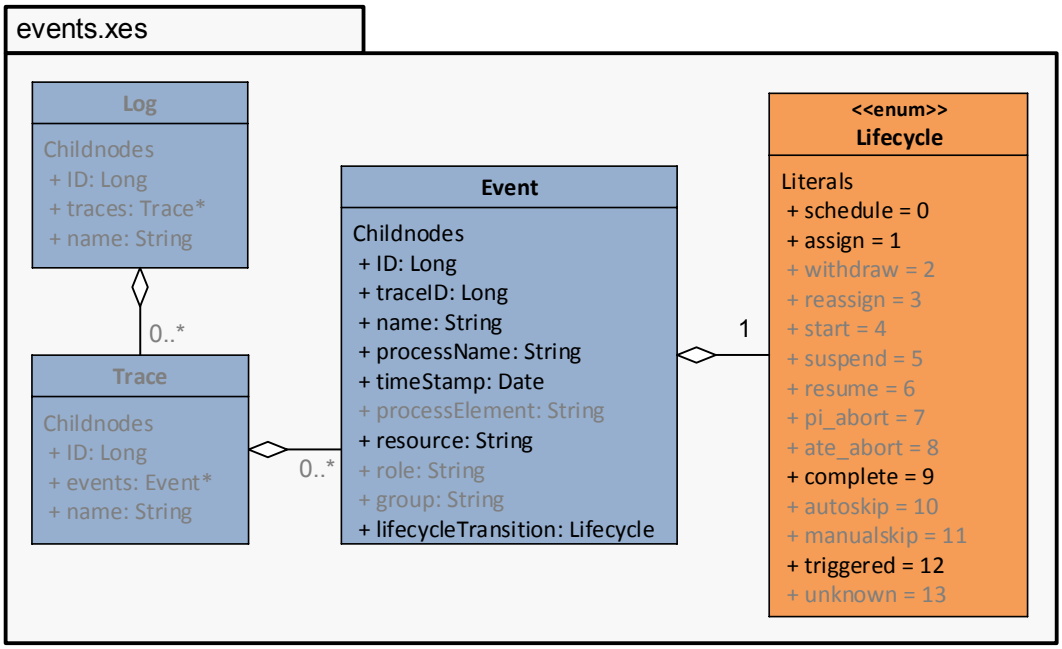


Fig. 5.8 Meta-Model for a BP Event Log (relevant elements are highlighted)

conforming to a general format capturing main features of the different source formats. In the case of the DPDF the global, standardised events conform to the prominent *eXtensible Event Stream* (XES) [90, 258] format which is also supported by existing process discovery tools such as ProM[12]. Figure 5.8 shows the meta-model for the default XES format including highlighted parts which are of relevance for the DPDF. Since the framework is a run-time solution that operates in a event-by-event basis the notion of *Log* is not supported and the notion of a *Trace* (record of an instance) is included in the attributes of an event (see *traceID*). Generally, the relevant elements can be mapped to the ones introduced in Section 2.2.2: *Event ID = ID, Timestamp = timeStamp, Instance ID = traceID, Name = name, Resource = resource*, and *Lifecycle = lifecycleTransition*. One element not introduced yet is the *processName* which is a unique identifier to map an occurring event to its business process. It is required since the DPDF monitors enterprise systems with multiple connected or not directly dependent processes. Note, that the DPDF only supports a simple lifecycle (including event types *schedule, assign, complete*, and *triggered*) since this is a part that is very difficult to standardise between the different formats. Events with other lifecycles have to be either discarded or (preferably) mapped by the responsible adapter in the *Event Hub*. The XES format was chosen to enable an extension of the format at a later point in order to allow for additional information to be observed and analysed, further supporting the Extensibility requirement TR6.

Event Hub The main task of the *Event Hub* is the translation of BPMS-specific events to events which conform to the standardised format. This is achieved via adapters which implement this translation. With regards to the translation the following problems had to

[12] http://www.processmining.org

be addressed: (1) Two BPMS might work in different timezones, which is why the timestamp value has to be translated to a unified timezone (e.g. UTC) in order to avoid analysis errors; (2) activity names might have different formats or might collide even though they originate from two different activities. In these cases a mapping to a unified naming system has to be in place for the various adapters. The same applies for other event information, e.g. the *Trace ID*: If two different BPMS use two differing trace ids for the same trace, the event hub has to map them while processing, e.g. by using an alternative event feature to create the mapping.

5.3.3 Dynamic Footprints

The *Dynamic Footprint* (see Figure 5.7) is the model that contains the state of the business process in an abstract form and acts both as an input and output of the run-time event processing. With each event the footprint is potentially updated. It has to be noted at this point again that the dynamic footprint model is only abstract information (in the form of vectors and matrices) and still has to be interpreted into a proper business process model. The term footprint has been borrowed from the process discovery terminology: In many process discovery algorithms it is common to first build a footprint which is then analysed and transformed into a business process model, e.g. α-algorithm [249], HeuristicsMiner [278], or CCM (see Section 5.2).

One of the main challenges for the framework was the design of the footprint: On one hand it has to be expressive enough to enable the translation into the different aspects of a business process state; on the other hand size constraints were to be met in order to ensure a quick update of the footprint during run-time. Furthermore, to support the requirement TR5 (optimised algorithmic run-time/memory usage) the size of the footprint has to be independent from the total number of occurred events and from the total number of occurred traces. This is necessary to keep the algorithmic run-time of processing one event constant. Only the number of activities and resources is influencing the size of some parts of the dynamic footprint which is explained later. Another finding during the development of the framework was that, apart from the *Open Traces*, all parts of the footprint should avoid absolute statements, i.e. true/1 and false/0, but instead use weights, e.g. statement A is true with a probability of 0.92 on a scale from 0 to 1. These statement weights can than be updated incrementally with each event, either supporting or opposing the statement. In accordance with TR6 (Extensibility) the footprints are modularly separated along the same dimensions as the DBPMRT: control-flow, performance, resources, instance states (*Open Traces*). In this way complete perspectives can be added and removed as required for individual reasoning purposes. The respective footprints are briefly described in the following:

Control-Flow Footprint contains the same information as the footprint of the CCM introduced in Section 5.2.3: (1) the two matrices *"appears before first"* ($x \rhd y$) and *"appears*

before" ($x \rhd \rhd y$) which contain abstracted information about the global relationship between activities. Their size is $n * n$ with n being the number of involved activities; and (2) Three additional vectors *"occurrence once"* ($Oon(x)$), *"occurrence overall"* ($Oov(x)$), and *"first element"* ($Fel(x)$) store the probability and loop behaviour of an individual activity. The length of these vectors is n. Note, that this footprint was defined for a complete event log. The adaptation to allow for event-wise processing while conforming to the original semantic of the matrices/vectors is discussed in Section 5.4. The dynamic control-flow footprint additionally contains the *"directly-follows"* relation in matrix form (size $n * n$) similar to that of the HeuristicsMiner (see Figure 2.17 on page 42). It is, however, not used for interpretation but for specific sub-footprint calculation to further shorten the algorithmic run-time of the involved components. The details about this are discussed in later sections.

Performance Footprint consists of generic, type-level performance parameters like *process instance occurrence* or *activity net working time* and are stored as a normal distribution function, i.e. mean and deviation values (see Section 2.5.1). The size of this footprint increases linearly with the number of activities in the process. An exception is the *path probabilities* for decisions: For them the footprint is the *directly-follows* matrix as introduced in the previous paragraph. The size of the footprint increases quadratically in relation to the number of involved activities. For the reasoning purpose of prediction via simulation the performance footprint is only focused on capturing "BP defining" parameters which are necessary to carry out a simulation, i.e. *process instance occurrence*, *activity net working time*, and *path probabilities*. If other parameters like *End-to-End Processing Time* should be part of the DBPMRT, e.g. for other reasoning purposes, the performance footprint and the associated adapter need to be extended. The formal details of the footprint calculation are defined in Section 5.4.3.

Resource Footprint consists of a matrix that associates each activity to a resource depending on the probability of a resource being assigned to perform this activity. The size of the matrix is $n * n_r$ with n being the number of activities and n_r the number of resources. The formal description of the resource footprint is presented in Section 5.4.2.

Open Traces consists of the last lifecycle transition of each activity that has appeared in each open trace. This is the only part of the footprint that consists of absolute statements on the instance level as opposed to heuristic abstractions on the type level. The details of this footprint and its extraction are discussed in Section 5.6.

5.3.4 Modified Methodology for the Control-flow Perspective: The Dynamic Construct Competition Miner

One of the most challenging parts for realising the DPDF is the implementation of the concept for the control-flow perspective. For this reason an overview of the modifications applied to the original CCM (see Section 5.2) in order to comply with the DPDF framework is provided in this separate (sub-)section. The result of these modifications is called Dynamic Construct Competition Miner (DCCM), an event-based methodology that discovers type level control-flow information and features the same general characteristics as the CCM with regards to autonomy, quality of results, and robustness (requirements TR1-TR3). Of particular interest for the transformation of the CCM algorithm to a solution for dynamic process discovery is the composition of the footprint and its calculation from the log: It is deterministic, based on heuristics (relative values), and on global relations between activities. Furthermore, not only one overall footprint is created by the CCM but also for each divide-and-conquer step two new subsets A_{first} and A_{second} are distinguished for which new sub-footprints have to be created (see Algorithm 1 on page 139).

Overview of the Modifications

The following modifications were applied to the default CCM to create the DCCM:

1. Splitting up the algorithm in two separate parts as proposed for the DPDF: one for dynamically updating the current footprint(s) complying to the run-time requirement TR5, and one for interpreting the footprint into a control-flow model which has less restrictions with regards to its execution-time.

2. In the CCM the footprint is calculated in relation to all occurring traces. This is not applicable for DPDF since the number of traces should not have an influence on the execution-time of any component of an DPDF solution. For this reason the footprint has to be calculated in a dynamic fashion, i.e. an event-wise footprint update independent from the previously occurred number of events or traces.

3. The original behaviour of the CCM to carry out a footprint calculation for every subset that has been created by the divide-and-conquer approach is not optimal since then the DCCM would have to extract up to $2 * n + 1$ different footprints if only one activity was split-up from the main set for each recursion[13]. This has been improved for the DCCM: for the most common constructs Choice and Sequence the sub-footprints are automatically derived from the parent footprint (with the help of the directly-follows relation that was added to the footprint).

[13] e.g. for $A = \{a,b,c,d\} : (a,b,c,d) \rightarrow ((a,b,c),(d)) \rightarrow (((a),(b,c)),(d)) \rightarrow (((a),((b),(c))),(d))$, seven different footprints for sets $\{a,b,c,d\},\{a,b,c\},\{b,c\},\{a\},\ \{b\},\{c\},\{d\}$ need to be created - (,) denote the nested blocks that emerge while splitting the sets recursively.

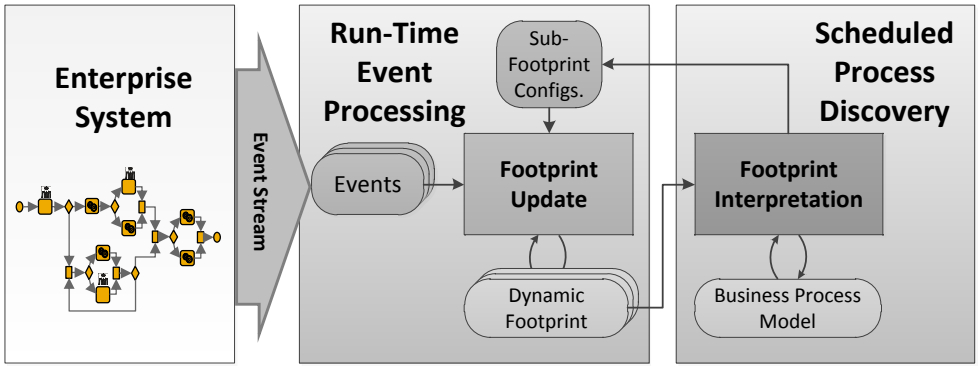


Fig. 5.9 Conceptual Methodology of the Dynamic CCM

4. In some use cases it was observed that for every occurring event the state of the process is alternating between a number of different control-flows. This is caused by "footprint equivalent" BP models, i.e. two models are footprint equivalent if they both express the behaviour captured by the footprint. To mitigate this a measure which favours the last control-flow state in order to prevent the described behaviour was introduced.

Details about the individual modifications are provided in later sections, i.e. modification 2 is described in Section 5.4.1, modifications 3 and 4 in Section 5.5.1, and modification 1 in the following (sub-)section.

Modified Methodology

The original CCM algorithm had to be split up into two separate parts. A component triggered by the occurrence of a new event to update the dynamic footprint and a component decoupled from the event processing which interprets the footprint into a BP Model. The conceptual methodology of the DCCM is depicted in Figure 5.9. The components, models, and functionality of the DCCM are described in the following: Events from the monitored *Enterprise System*, in which the end-to-end process is deployed, are fed into an event stream. The *Footprint Update* component is the receiver of these events and processes them directly into changes on the overall *Dynamic Footprint* which represents the abstract state of the monitored business process. If additional footprints for subsets of activities are required as specified by the *Sub-Footprint Configurations*, e.g. if a Loop or Parallel construct was identified, then these sub-footprints are also updated (or created if they were not existent before). The *Dynamic Footprint(s)* can then at any point in time be compiled to a human-oriented representation of the BP control-flow by the *Footprint Interpretation* component, i.e. the abstract footprint representation is interpreted into knowledge conforming to a block-structured BP model. In the DCCM this interpretation is scheduled dependent on how many new completed traces have appeared since the last interpretation, e.g. the footprint interpretation is executed once every 10 terminated traces. If the *interpretation frequency* $m \in \mathbb{N}$ of the DCCM is set to 1 a footprint interpretation is executed for every single trace that terminated. The *Footprint Interpre-*

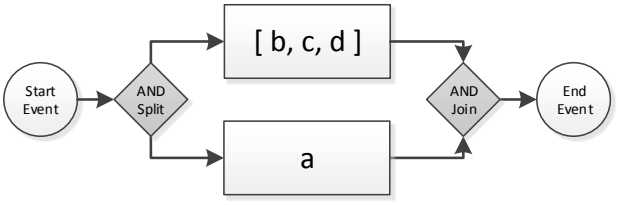


Fig. 5.10 Result of the *Footprint Interpretation* on an event stream produced by the example from Figure 5.3 if no sub-footprint for $\{b,c,d\}$ is available yet - only the top-level split has been discovered and due to the missing sub-footprint one activity $[b,c,d]$ is created

tation algorithm works similar to the CCM algorithm shown in Algorithm 1 on page 139 but instead of extracting footprints from a log (line 8), the modified algorithm requests the readily available *Dynamic Footprint(s)*. If a sub-footprint is not yet available (e.g. at the beginning or if the process changed) the *Footprint Interpretation* specifies the request for a sub-footprint in the *Sub-Footprint Configurations* in the fashion of a feedback loop. Thus, *Sub-Footprint Configurations* and *Dynamic Footprints* act as interfaces between the two components, *Footprint Update* and *Footprint Interpretation*. A sub-footprint configuration defines how the event stream is monitored to create the sub-footprints and consists of the three activity sets A_m, A_i, and A_t (activities to monitor, ignore, or tolerate) which were introduced in Section 5.2.3. If no sub-footprint for these exist yet the *Footprint Interpretation* cannot continue to analyse the subsets. In this case, usually occurring in the warm-up or transition phase, an intermediate BP model is created with activities containing all elements of the unresolved sets as exemplary depicted in Figure 5.10.

5.3.5 Description of other Framework Artefacts

The remaining general components are briefly described in the following to help understanding the DPDF's overall concept. A detailed and more formal description of these components is provided in later sections or chapters.

Run-time Processing of Standardised Events

As shown in Figure 5.7 the *Run-time Processing of Standardised Events* agent processes the global standardised events to updates in the dynamic footprint and thus updating the abstract state of the business process. Each of the agents/methods included in this component should be deterministic and designed as independent as possible, i.e. without any input from other perspectives. The *Control-flow Footprint Update* as well as the *Trace State Monitoring* agents are to some extent an exception to this constraint which is discussed at a later stage. Furthermore, each processing agent should only take a constant amount of time, independent from the total number of previously occurred events or traces. The total number of involved activities can have, however, a linear increase

of the run-time due to recalculating the relation of the occurred activity to, in the worst case, all other activities. Overall the agent has to scale linearly to the number of events occurring. Since the event stream is to be processed sequentially due to the incremental methods applied the framework can be scaled up easily by adding computational power.

Essentially, the main challenge was to split up the process discovery algorithms into a footprint update part and a footprint interpretation part. Additionally, the footprint update part had to be dynamic, i.e. work incrementally and always update the current dynamic footprint in a way that older events have less influence than the newer events. The approach utilised in the DPDF framework is based on the ageing principle explained in detail in Section 5.4. Note, that the Trace State Monitoring is not updated in a dynamic way since it consists of absolute rather than heuristic values. Instead only the lifecycle transition for the activity and trace associated with the occurred event is updated (see Section 5.6). Also note, that if any information is not available from the events, some parts of the footprint may neither be discovered nor updated, e.g. if the resource information is not provided the resource footprint cannot be build, or if the lifecycle transitions were not to be provided, the performance footprint in general and the activity net working time in particular could not be properly discovered without further information about the control-flow of the business process.

Footprint Interpretation

The *Footprint Interpretation* (see Figure 5.7) is the agent that translates the abstract state information in form of the dynamic footprint into state information that conforms to business process notations of a certain purpose, i.e. a DBPMRT. As opposed to the run-time event processing agent it has less restrictive constraints with regards to run-time as it is only executed on demand, after a specified amount of time passed or after a specified number of events or traces occurred. However, since this framework is designed to enable real-time or near real-time analysis the run-time should not increase exponentially in relation to the number of activities involved.

Generally, the footprint interpretation is required to have exchangeable adapters since different requirements on the state representation may need to be met. For instance, if only performance information is to be monitored (similar to BAM solutions) it is sufficient to only discover the associated activities but not the complete control-flow. Another challenging problem that had to be addressed during development was that for some footprints multiple interpretations existed, i.e. two different control-flows can produce the exact same traces. In such cases it could happen that the *state of the business process* was alternating between these options which is an undesired behaviour. To prevent this from happening in the DPDF framework, a technique is applied that favours the last control-flow in order to prevent the alternation between different but trace-equivalent versions. The details of the agents involved in the footprint interpretation are presented in Section 5.5.

Current State of Business Process

The current state of the business process(es) is represented by the DBPMRT specified in Section 4.2. When adding a time dimension to the specification and not discarding older states (of the type level), the DBPMRT may also represent the evolution of the monitored BPs as discussed in Section 4.3.

Reasoning on DBPMRT

The DBPMRT (as state or evolution model) may be used for a certain reasoning purpose (or simply be displayed and allow for expert reasoning). In Chapter 6 the use case of predicting the development of PPIs via simulation based on the DBPMRT is presented and serves as evaluation of the DPDF and the applied concepts.

5.3.6 Computer- vs. Human-oriented Run-time Models

The main concept proposes the introduction of the dynamic footprint as intermediate abstraction level. The dynamic footprint matrices and vectors are essentially aggregated and heuristic type level abstractions of the occurring events/traces which are, however, not residing on the same level of abstraction as common imperative BP standards. Only via interpretation algorithms that are more run-time cost intensive the information on BP level, i.e. the DBPMRT, can be inferred. The result is that the DPDF's concept maintains information on two different levels. Both of which can be considered run-time models:

1. The dynamic footprint represents a *Computer-oriented Run-time Model* not conforming to a human-interpretable model language but with a direct causal link to the system, i.e. changes in the system are quasi-immediately reflected in the footprint.

2. On the other hand the DBPMRT represents a *Human-oriented Run-time Model* conforming to a human-interpretable model language which, however, features a less direct link to system due to the separated interpretation lifecycle, i.e. changes in the system may be reflected in the DBPMRT after a potential delay.

The timeliness of a change reflection is different for the human-oriented run-time model than for the computer-oriented run-time model since it may not be practical to execute the interpretation for every event in cases of big but fast processes. For "slow" processes, e.g. one event per second, this might be entirely reasonable, however, this is not applicable when dealing with systems potentially producing thousands of events per second. One way to mitigate this is to introduce thresholds that trigger a re-interpretation for perspectives where the interpretation is not combinatoric-based[14] but the inferred entities

[14]For instance, the competition algorithm of the CCM (see Section 5.2.5) is combinatoric-based since it needs to find the best combination for suitable construct and the members of two distinctive sets of activities.

and relations in the DBPMRT are directly related to values in the footprint: For instance, the relations between activities, roles, and resources are directly based on whether or not a footprint value has crossed a certain threshold. If such a crossing of a threshold is observed a re-interpretation of this part of the respective BP perspective can be triggered. If this approach is followed the interpretation is again directly connected to the event-lifecycle rather than being in a separate lifecycle. In order to support the greatest possible extent of a timely causal link from system to DBPMRT, the DPDF follows this approach for BP perspectives when applicable (e.g. see Section 5.5.2). Note, that this is not applicable for the control-flow perspective since the associated interpretation is combinatoric-based. As a result, the DPDF can only achieve a direct causal link between system and DBPMRT if the use case is "slow" (interpretation at every event) or based on BPs with fixed control-flows (no control-flow interpretation required). Since the objective of this thesis is to deal with large-scale, dynamic enterprise systems (see particularly requirements TR4 and TR5) the control-flow interpretation remains a separate lifecycle and thus represents a less direct causal link. In the DCCM evaluation section 5.8.2 it is investigated if and to which extent this "weak" causal link is of significance.

In conclusion, this section presented the general concept of the DPDF and a specification of its components. The goal of the DPDF framework is to maintain the causal link from an enterprise system to a DBPMRT in a volatile run-time environment, i.e. meeting the requirements defined for such a framework in Section 5.1. The main feature of the concept is that the inference of high-level BP information from low-level event data was divided into two lifecycles, *Event Processing* (for each event) and *Footprint Interpretation* (periodically executed). While some of the concept's artefacts like *Event Hub*, the *Global, Standardised Events*, the *Dynamic Footprint*, as well as the updated methodology of the CCM towards the DCCM were discussed in detail, the features and functionality of the other components were only briefly introduced to support the understanding of the overall concept. However, detailed information of the components for footprint update and interpretation as well as the state tracking on the instance level are presented in the following.

5.4 Dynamic Footprint Update

The *Dynamic Footprint Update* components process *Global, Standardised Events* to changes in the *Dynamic Footprint*, i.e. updates the abstract representation of the process state (type and instance level). In this section the specifics of the update components for the footprints of the type level BP perspectives, i.e. control-flow, resources, and performance, are presented.

5.4.1 Control-Flow Footprint

The modifications to the original CCM algorithm are threefold: (1) the introduction of the dynamic concept of ageing (which is then also used for the other perspectives), (2) the adding and removal of activities to the (control-flow) footprint that appear or disappear from the event stream, and (3) the introduction of the directly-follows relation into the footprint to decrease the required interpretation time.

The Concept of Ageing

The original footprint extraction of the CCM algorithm calculates all values in relation to the number of occurred traces, i.e. every trace's influence on the footprint is equal: $\frac{1}{|\Lambda|}$ ($|\Lambda|$ is notation for number of traces in log Λ). To comply to the algorithmic run-time requirement TR5 the footprint update calculation should only take a fixed amount of time, independent from the total number of previously occurred events or traces. An increase of the total number of involved activities can cause, however, a linear increase of the execution-time due to the recalculation of the relations between the occurred activity and, in the worst case, all other activities. The independence from previous traces is the reason the footprint is calculated in a dynamic fashion, i.e. the dynamic footprint is incrementally updated in a way that older traces "age" and thus have less influence than more recent traces. Note, that the dynamic footprint is not updated for each event but for each finished trace. This is necessary since the footprint consists of global relationships between activities, for which reliable statements can only be made for completed traces.

The general ageing approach that is utilised in the footprint update of the DCCM is based on the calculation of an individual *trace footprint*[15] (*TFP*) for each trace which influences the dynamic overall footprint (*DFP*). For the n-th new TFP_n the *DFP* is updated in the following way: Given a specified *trace influence factor* $t_{if} \in \mathbb{R}$ with $0 < t_{if} \leq 1$ the old DFP_{n-1} is aged by the *ageing factor* $a_f = 1 - t_{if}$, i.e.

$$\begin{aligned} DFP_n = \quad & t_{if} * TFP_n + \quad (1 - t_{if}) * DFP_{n-1} \\ \text{or alternatively} \quad DFP_n = \quad & (1 - a_f) * TFP_n + \quad a_f * DFP_{n-1} \end{aligned} \tag{5.8}$$

E.g., for trace influence factor $t_{if} = 0.01$: $DFP_n = 0.01 * TFP_n + 0.99 * DFP_{n-1}$. An example of this footprint ageing is presented in Appendix Chapter B where the new footprint DFP_{35} is calculated based on the new trace footprint TFP_{35} from trace $[b, a, c, a, b, c, b, a, d, e, e, d]$ and the old footprint DFP_{34} (which is equivalent to the example footprint $FP^{L_1, A}_{\{\},\{\}}$ from page 144).

Two different specialised ageing methods have been developed which are specified in the following and evaluated against each other in Section 5.8.2: *Discrete Ageing* and *Time-dependent Ageing*.

[15] the occurrence values for activities as well as the global relations (see Definition 7 in Section 5.2.3) are represented in the trace footprint by absolute statements *true* $\equiv 1$ if the relation is true and *false* $\equiv 0$ if not

Discrete Ageing is the ageing method based on the occurrence of a new trace. That means, that the trace influence t_{if} is a fixed value and the *DFP* ages the same proportion for every time a trace footprint is added indeterminate from how much time has passed since the last footprint update. For example, assuming $t_{if} = 0.01$ the trace footprint TFP_n has the influence of 0.01 when it first occurs (see Equation 5.8); after another TFP_{n+1} has occurred the influence of TFP_n decreases to $0.01 * 0.99$, and after another $0.01 * 0.99^2$ and so on. By applying this incremental method, older *TFP* are losing influence in the overall dynamic footprint. Figure 5.11 shows how the influence of a trace is dependent on its "age": If $t_{if} = 0.1$, the influence of a trace that appeared 60 traces ago became almost irrelevant. At the same time if $t_{if} = 0.01$ the influence of a trace of the same age is still a little more than half of its initial influence when it first appeared. Essentially, the purpose of the *trace influence factor* t_{if} is to configure both, the "memory" and the adaptation rate, of the footprint update component, i.e. a high t_{if} means quick adaptation but short memory but a low t_{if} means a slow adaptation but a long memory. Finding the correct trace influence is an issue of balancing these two inversely proportional effects, e.g. it might be generally desirable to have a high adaptation rate ($t_{if} = 0.1$) but if certain behaviour of the process only occurs once in every 60 traces it will already be "forgotten" when it reappears (see Figure 5.11) essentially resulting in a continuously alternating business process.

However, while applying this method it was observed that at the beginning of the event streaming an unnecessarily long time to "warm-up" was required until the *DFP* reflected the correct behaviour of the business process. The reason for delay is that the first dynamic footprint DFP_0 starts with matrices and vectors containing only 0s and, depending on the influence factor, it might take a large number of traces until the influence of DFP_0 disappears. In order to shorten the "warm-up" phase of the *Footprint Update* a more dynamic method was adopted: If the overall number of so far occurred traces $n - 1 < \frac{1}{t_{if}}$ then the influence of the dynamic overall footprint DFP_{n-1} is $\frac{n-1}{n}$ and of the new trace footprint TFP_n is $\frac{1}{n}$. As a result all traces that occur while $n - 1 < \frac{1}{t_{if}}$ have the same influence in the DFP_n of $\frac{1}{n-1}$. For instance if $t_{if} = 0.01$ and $n = 10$ then a new dynamic footprint is calculated with $DFP_{10} = \frac{1}{10} * TFP_{10} + \frac{9}{10} * DFP_9$ and for the next trace $DFP_{11} = \frac{1}{11} * TFP_{11} + \frac{10}{11} * DFP_{10}$ and so on. As soon as $n - 1 \geq \frac{1}{t_{if}}$ the standard discrete ageing with a fixed influence factor is adopted:

$$DFP_n = \begin{cases} \frac{1}{n} * TFP_n + \frac{n-1}{n} * DFP_{n-1} & \text{if } n < \frac{1}{t_{if}} + 1 \\ t_{if} * TFP_n + (1 - t_{if}) * DFP_{n-1} & \text{if } n \geq \frac{1}{t_{if}} + 1 \end{cases} \tag{5.9}$$

With the help of this additional modification the "warm-up" phase of the *Footprint Update* could be drastically reduced, i.e. processes were already completely discovered a few traces after the start of the monitoring which will be shown in the evaluation Section 5.8.2.

Time-dependent Ageing is an ageing method based on the time that has passed since the last trace occurred. The more time has passed the less influence the old DFP_{n-1} has

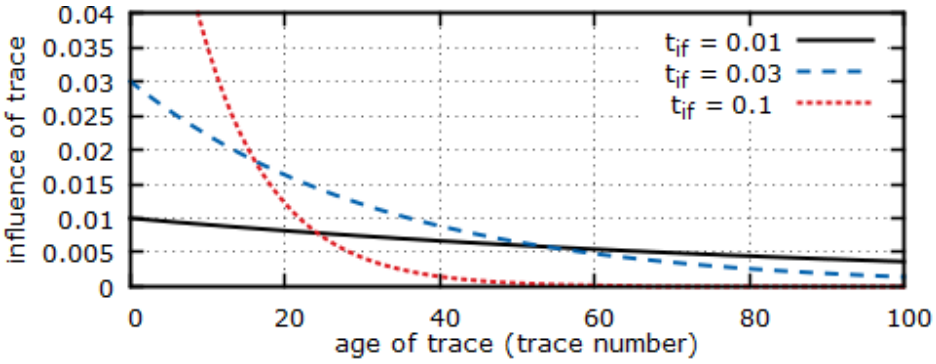


Fig. 5.11 Development of the influence of a trace for different trace influence factors (t_{if})

on the updated DFP_n. This is achieved in a similar way than in the discrete ageing but instead of having an ageing factor a_f that is fixed or relative to the number of occurred traces (see Equation 5.9) it is now relative to the time passed. That means that ageing factor a_f and trace influence factor t_{if} are now calculated based on an *ageing rate* a_r per passed *time unit* t_{ur}, i.e. in particular time t_n has passed since the last completed trace occurred then:

$$a_f = a_r^{\frac{t_n}{t_{ur}}} \tag{5.10}$$

If $t_n = t_{ur}$ then the dynamic overall footprint ages exactly the same as with the discrete ageing, if $t_n > t_{ur}$ then it ages quicker, and if $t_n < t_{ur}$ it ages slower. For instance, with ageing rate $a_r = 0.99$ and time unit $t_{ur} = 1s$: If the new trace occurred $t_n = 2s$ after the last footprint update then the new dynamic overall footprint $DFP_n = (1-0.99^2) * TFP_n + 0.99^2 * DFP_{n-1}$. Through time-dependent ageing the influence development of passed traces behaves similarly to the discrete ageing in Figure 5.11 apart from that the ageing is not based on the number of traces but on the actual time passed (the numbers on the x-axis now represent the passed time units since the trace occurred). For the time-dependent ageing a similar problem was observed during the warm-up phase as with the discrete ageing: Since the *DFP* only consists of zeros when initialised it takes an unnecessary long time to converge towards a footprint representing the correct behaviour of the business process. For this reason an alternative linear ageing, relative to the overall passed time since the first trace recorded t_{ur}, was adopted. The final ageing factor a_f is the minimum of both calculated values as shown in Equation 5.11.

$$a_f = \min(1 - \frac{t_n}{t_{all}}, a_r^{\frac{t_n}{t_{ur}}}) \tag{5.11}$$

Considering the example from earlier where $a_r = 0.99$, time unit $t_{ur} = 1s$, the new trace TFP_n occurred $2s$ after the last update, and the first trace recorded was $t_{all} = 4s$ then $a_f = \min(1 - \frac{2s}{4s}, 0.99^2) = 0.5$ and according to Equation 5.8: $DFP_n = (1-0.5) * TFP_n + 0.5 * DFP_{n-1}$. In this way the warm-up phase can be shortened similar to the discrete approach but is still be based on time.

Appearing or Disappearing Activities

Another important dynamism feature that had to be implemented was the possibility to add an activity that has not appeared before. A new activity is first recorded in the respective trace footprint. When the trace is terminated it will be added to the overall footprint in which it is not contained yet. The factored summation of both footprints to build the new dynamic footprint is carried out by assuming that a relation not previously in the dynamic overall footprint contained has a value 0. As a result, it might take a number of traces that contain the new activity until the correct behaviour is captured in the footprint.

Furthermore, activities that do not appear any more during operation should be removed from the dynamic footprint. This was implemented in the DCCM in the following way: If the *occurrence once* value $Oon(x)$ of an activity x drops below a removal threshold $t_r \in \mathbb{R}, t_r < t_{if}$ it is removed from the dynamic footprint, i.e. all values and relations to other activities are discarded.

Garbage Collection for Sub-Footprint Requests

The discarding of previously requested sub-footprints (see Section 5.3.4: Modified Methodology) which have not been requested for a certain number of footprint interpretations works in a similar fashion. The default configuration of the DCCM is that after 10000 interpretations without requesting a particular sub-footprint it will be removed from the set of footprints to monitor.

The Directly-Follows Relation

The fact that especially many Choice and Sequence constructs are present in common business processes, motivates an automated sub-footprint creation in the *Footprint Interpretation* based on the parent footprint rather then creating the sub-footprint from the event stream. This step helps to decrease the execution-time of the *Footprint Update* by reducing the numbers of sub-footprints the component needs to keep track of[16]. It is achieved by introducing an extra relation to the footprint - the *directly follows* relation as used by other mining algorithms (see Section 2.4.2).

Definition 12 *Given a (sub-)trace $\lambda \in A_m^*$ then the directly follows relation for two activities $x, y \in A_m$ is:*

$$|x \gg y| = |\{i \in \{0,1,...,|\lambda|-2\} \mid \lambda(i) = x \wedge \lambda(i+1) = y\}|$$

Examples for trace $[b,a,c,a,b,c,b,a,d,e,e,d]$: a is directly followed by b once, i.e. $a \gg b = 1$; b is directly followed by a twice, i.e. $b \gg a = 2$; d is never directly followed by c, i.e. $d \gg c = 0$. This relation is part of the footprint in matrix form and updated

[16]Note, that rare cases (if Loop and Parallel constructs dominate) this modification can have a negative effect on the execution-time since extra information needs to be extracted without the benefit of mining less sub-footprints

in the same way as introduced earlier. Note, that the values can become greater than 1. In the *Footprint Interpretation* this relation is then used for creating the respective sub-footprints for Sequence and Choice constructs but not for identifying BP constructs since the *directly follows* relation does not represent a global relation between activities (see Section 5.5.1).

5.4.2 Resources Footprint

The resources footprint is an abstraction from event information that is used for interpretation of the resource perspective. For the update of this footprint only the events that record the assigning of a resource in order to execute an activity are of relevance, i.e. events are filtered for the lifecycle "assign" (see Figure 5.8). Furthermore, unlike the update for the control-flow footprint, the resource footprint is updated with every event (of the "assign" lifecycle). Note, that only one overall resource footprint exists, i.e. no feedback loop or sub-footprints are part of the concept.

For the definition of the resource footprint events are not regarded as elements of a *simple trace* or *simple event log* (see Section 5.2.1), i.e. only represent the associated activity or BP element, but are instead tuples (x, r) with activity $x \in A$ and resource $r \in R$:

Definition 13 *Let Λ_{res} be the multi-set[17] of tuples $(x, r) \in A \times R$ representing all occurred events (with the lifecycle "assign"), then is*

$$x \otimes^{\Lambda_{res}} r \quad = \quad \frac{|\{(x_1, r_1) \in \Lambda_{res} \mid x_1 = x \wedge r_1 = r\}|}{|\{(x_1, r_1) \in \Lambda_{res} \mid x_1 = x\}|}$$

The resource footprint consists of heuristic relations between activities and their assigned resources $x \otimes^{\Lambda_{res}} r$ that represent the respective probability of resource r being assigned to activity x in relation to all resource assignments to activity x in Λ_{res}. Consider, for instance, $\Lambda_{res} = \{(a, r_1)^2, (a, r_2)^3, (b, r_2)^2, (b, r_3)^2\}$ then the resulting resource footprint FP^{res} is

$$\begin{array}{c} \\ a \otimes^{\Lambda_{res}} r: \\ b \otimes^{\Lambda_{res}} r: \end{array} \begin{array}{c} \begin{array}{ccc} r_1 & r_2 & r_3 \end{array} \\ \begin{pmatrix} 0.4 & 0.6 & 0 \\ 0 & 0.5 & 0.5 \end{pmatrix} \end{array}$$

Note, that the sum of all probabilities of an activity is 1, i.e. $\forall_{x \in A} : \sum_{r \in R} (x \otimes^{\Lambda_{res}} r) = 1$.

Definition 13 defines the probabilities in the resource footprint in absolute terms for a complete (multi-)set of occurred events. Similarly as with the control-flow footprint, the ageing concept (discrete and time-dependent) can be applied to the resource footprint in order to facilitate for a dynamic setting where older events are to lose influence over time. If the above footprint is considered to be the current dynamic overall footprint for resources DFP^{res}_{n-1} and a new event (a, r_1) occurs, then is with a trace influence of $t_{if} = 0.1$ the new $a \otimes r$ relation updated to $0.1 * (1 \quad 0 \quad 0) + 0.9 * (0.4 \quad 0.6 \quad 0) = (0.46 \quad 0.54 \quad 0)$ and the new dynamic overall resource footprint DFP^{res}_{n} is updated to

[17] Multiples tuple entries possible

$$\begin{array}{c} \\ a \otimes r: \\ b \otimes r: \end{array} \begin{array}{c} \begin{array}{ccc} r_1 & r_2 & r_3 \end{array} \\ \begin{pmatrix} 0.46 & 0.54 & 0 \\ 0 & 0.5 & 0.5 \end{pmatrix} \end{array}$$

for which still the condition applies that the sum of the probabilities for each activity is 1.

For each resource footprint update the resulting values are tested whether or not they crossed (in either direction) a certain tolerance threshold. If this is the case a new resource interpretation is triggered which directly updates the DBPMRT's resource perspective. The details of this threshold and why it is indicating a model change are further discussed in Section 5.5.2. Furthermore, this component features a activity or resource removal policy similar to the one described in the control-flow footprint update component.

5.4.3 Performance Footprint

The performance footprint is an abstraction from event information that is used for interpretation of the BP performance values required for BP simulation, i.e. *activity net working times*, *process instance occurrences*, and *path probabilities*.

The two performance values activity net working time and process instance occurrence are recorded as durations, i.e. the process instance occurrence is the time passed between two occurring BP instances; and the activity net working time is the duration from (1) the assignment of a resource to the activity until its completion (for human activities) or (2) the scheduling of an activity execution to its completion (for automated activities). Furthermore, these values which in reality are subject to a varying degree of fluctuation are treated as probabilistic values, i.e. with mean and variance, in order to better reflect reality.

Definition 14 *Let $T = \{t_1, t_2, ..., t_N\}$ be a multi-set of size N with values $t_1, t_2, ..., t_N \in \mathbb{R}$, then T can be represented by a tuple $v^T = (mean^T, var^T)$ with*

- *the average/mean over T: $mean^T = \frac{1}{N} * \sum_{i=1}^{N} t_i$, and*
- *the variance over T: $var^T = \frac{1}{N} * \sum_{i=1}^{N} (mean^T - t_i)^2$.*

The example $T_1 = \{1,3,4,2,3,5\}$ can be represented by $mean^{T_1} = \frac{1+3+4+2+3+5}{6} = 3$ and $var^{T_1} = \frac{(3-1)^2+(3-3)^2+(3-4)^2+(3-2)^2+(3-3)^2+(3-5)^2}{6} = \frac{4+0+1+1+0+4}{6} = \frac{10}{6} = 1.4$, i.e. $v^T = (\,3,\ 1.4\,)$.

Definition 14 specifies the mean and variance for a (multi-)set of values of the same context (e.g. net working time for activity a) with equal weights for all elements. However, in a dynamic environment newer elements should have a higher influence than older elements. For this reason, the same incremental ageing concept as introduced earlier is applied in this case. E.g. for a new value $t_n = 5$ and the current mean value $mean_{n-1} = 3$, variance value $var_{n-1} = 1.4$ the updated mean and variance are for $t_{if} = 0.1$: $mean_n = 0.1 * 5 + 0.9 * 3 = 3.2$ and $var_n = 0.1 * (3.2-5)^2 + 0.9 * 1.4 = 0.324 + 1.26 = 1.584$, i.e. $v_n = (\,3.2,\ 1.584\,)$.

The overall dynamic footprint DFP_n^{perf} contains the activity net working time for each occurring activity and the process instance occurrence for each business process in the form of such a probability value. Additionally, the directly-follows relation as explained in Section 5.4.1 is part of the footprint to enable the computation of specific path probabilities by the performance interpretation (see Section 5.5.3). Note, that the general concept technically requires the directly-follows relation to be calculated twice due to the modular concept of the DPDF. However, in the implementation only one overall directly-follows relation is computed to avoid duplicated effort.

5.5 Dynamic Footprint Interpretation

The *Footprint Interpretation* (see Figure 5.7 on page 153) is the agent that translates the abstract state information in form of the dynamic footprint into the DBPMRT. As a result of the different interpretation components the respective perspectives are replaced (if changed) as well as associations to other perspectives are updated. Depending on the type of DBPMRT (state vs. evolution; what purpose is supported, e.g. BP simulation vs. performance monitoring) the exact implementation of updating the model accordingly might vary which is, however, outside of the scope of this thesis. This section is particularly concerned with the formal process of extracting human-oriented model information (on the type level) from the computer-oriented footprint information for the BP perspectives control-flow, resource, and performance, respectively and independently.

5.5.1 Control-Flow

The DPDF's dynamic footprint interpretation of the control-flow is part of the DCCM (methodology introduced in Section 5.3.4) which is an extension of the CCM modified to work in dynamic run-time environments. It consists of two parts: the *constructs suitability* computation discussed in Section 5.2.4 and in the case of a construct for multiple activities the *construct competition* algorithm discussed in Section 5.2.5. Additionally, for the dynamic interpretation two modifications have been implemented and are explained in the following:

Automated Sub-Footprint Creation

As discussed in Section 5.3.4, the original behaviour of the CCM (see Section 5.2.2) to retrieve a sub-footprint for each subset that has been created by the divide-and-conquer approach is not optimal. This is why, in the *Footprint Interpretation* the DCCM calculates the sub-footprints for the most common constructs, Choice and Sequence, from the available parent footprint:

1. For the Choice construct the probability of the exclusive paths are calculated with $P_{first} = \sum_{x \in A_{first}} Fel(x)$ and $P_{second} = \sum_{x \in A_{second}} Fel(x)$ with $Fel(x)$ being the occurrences

of x as first element (see CCM footprint description in Section 5.2.3). Then the relevant values of the parent footprint are copied into their respective new sub-footprints and normalised, i.e. multiplied with $\frac{1}{P_{first}}$ and $\frac{1}{P_{second}}$, respectively.

2. The sub-footprints for the Sequence construct are similarly built, but without the normalisation. Instead, the directly-follows relation is used to calculate the new overall probabilities of the sub-footprints which indicate whether or not each part of the sequence is optional.

Avoiding Alternating Control-flow Interpretations

If two or more BP constructs are almost identically suitable for one and the same footprint, a slight change of the dynamic footprint might result in a differently discovered BP. This may cause an alternating behaviour for the footprint interpretation, i.e. with almost every footprint update the result of the interpretation changes. This is undesirable behaviour which is why the competition algorithm was additionally modified as follows: All combinations of BP construct and subsets are by default penalised by a very small value, e.g. $\frac{t_{if}}{10}$, with the exception of the combination corresponding to the previously discovered BP model, hence reducing the risk of discovering alternating BP models.

5.5.2 Roles and Resources

The resource footprint interpretation constructs the resource perspective of the DBPMRT from the current resource footprint DFP^{res} defined in Section 5.4.2. The interpretation method is similar to clustering techniques from the data mining area and is conceptually depicted for an example resource footprint in Figure 5.12:

Filtering the Footprint: In a first step the footprint is filtered for all relevant activity-resource associations. An association is considered relevant if it crosses a certain activity-dependent threshold $t_{res}(x)$ with $x \in A$ which is relative to the number of relations $x \otimes r > 0$ in the footprint DFP^{res}, i.e. for a given *resource threshold* $T_{res} \in \mathbb{R}^{+}_{<1}$

$$t_{res}(x) = \frac{T_{res}}{|\{(x,r) \mid x \otimes r > 0\}|}.$$

Considering the example shown in Figure 5.12, the individual thresholds are $t_{res}(a) = t_{res}(b) = \frac{0.1}{3} = 0.033$, $t_{res}(c) = \frac{0.1}{4} = 0.025$, and $t_{res}(d) = \frac{0.1}{2} = 0.05$ (top, right in the figure). The relative threshold became necessary due to observed behaviour in the use cases: The resources-activity associations were often unbalanced, e.g. some activities had connections to 30 resources whereas others to only 3; This resulted in probabilities that were on average different by the factor 10 which makes the balanced application of a fixed resource threshold impossible.

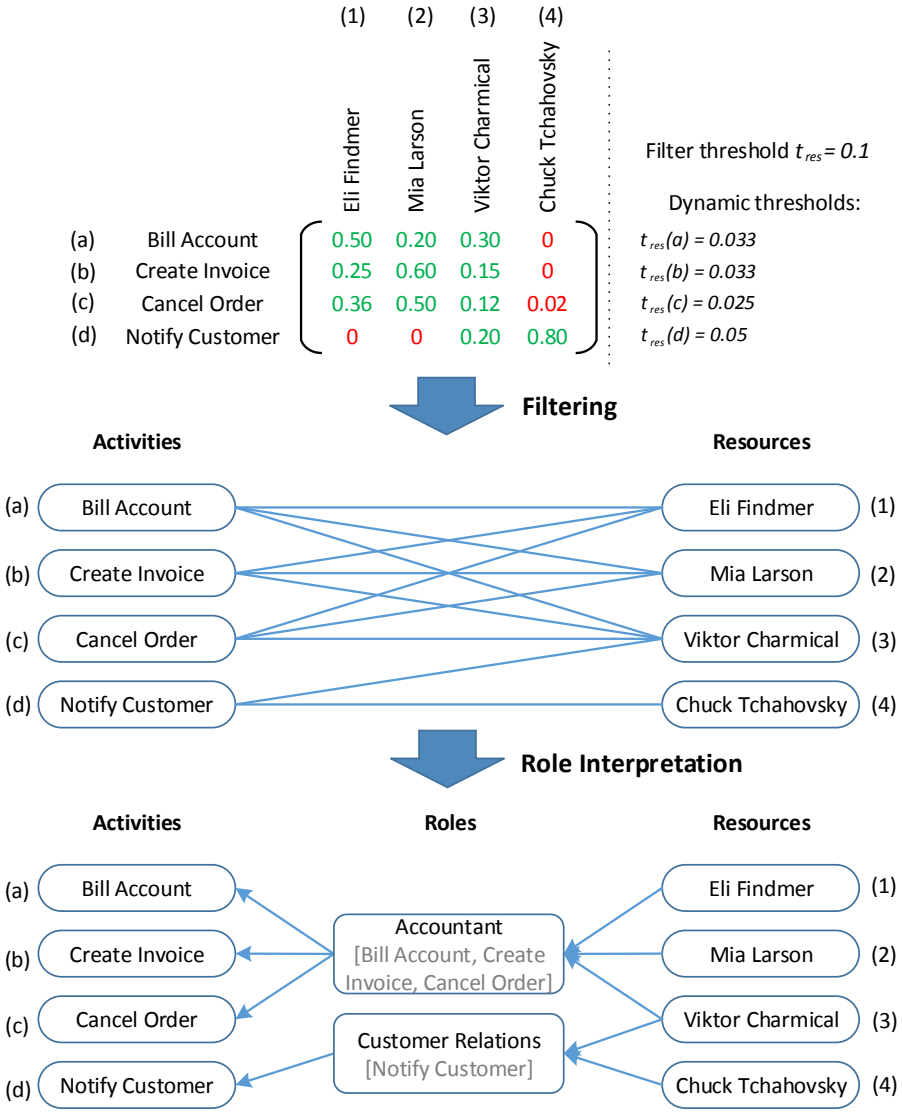


Fig. 5.12 Concept of the Role Discovery Shown on an Example

Based on the activity-related resource thresholds $t_{res}(x)$ the filtered activity-resource relation AR is calculated by

$$AR = \{(x,r) \mid x \otimes r > t_{res}(x)\}.$$

With regards to the example shown in Figure 5.12, the condition is true for the green highlighted elements of the matrix at the top. The constructed filtered relation contains the elements as shown in graph form in the middle of the picture: $AR = \{(a,1),(a,2),(a,3),(b,1),(b,2),(b,3),(c,1),(c,2),(c,3),(d,3),(d,4)\}$. Note, that the threshold checking is also carried directly after each update of the footprint values. When a threshold is crossed, it means that AB would have (at least) one relation more or less which again results in a different interpretation which is why the resource interpretation is triggered in that case. Moving the threshold checking into the update component makes the dynamic resource discovery methodology a direct (i.e. not in separate lifecycles) causal connection between system and the resource perspective of the DBPMRT.

Role Interpretation: In the final step the filtered activity-resource relation is processed to information in the resource perspective, i.e. (1) all (actively) involved resources as well as (2) the roles that represent the connection to the associated activities. Roles are an abstract concept not recorded by many BPMS, i.e. in reality the role field in the events is often empty or simply equals the resource name (i.e. each resource has its own unique role). For this reason this information is not taken into account by the DPDF (see Figure 5.8) but instead responsibilities (with regards to each resource) are grouped to represent the concept of the resources' roles. For this the map from activity $a \in A$ to a set of all associated resources $res(x)$ is defined as follows:

$$res(x) = \{r \mid (x, r) \in AR\}$$

This means for the above example that $res(a) = res(b) = res(c) = \{1,2,3\}$ and $res(d) = \{3,4\}$. In the next step all different mapping targets for all activities ($\bigcup_{x \in A} res(x) = \{\{1,2,3\}\{3,4\}\}$) are mapped to their associated activities:

$$RA = \{(R_a, A_r) \mid R_a \in \bigcup_{x \in A} res(x) \wedge A_r = \{x \in A \mid res(x) = R_a\}\}$$

For the example this means that $RA = \{(\{1,2,3\},\{a,b,c\}),(\{3,4\},\{d\})\}$. This set contains the representatives of roles (resource groups that are associated with the same activities) and the associated activities and resources. In the example these discovered roles are named "Accountant" and "Public Relations" (see bottom of Figure 5.12) to improve the understanding. This context information is, however, not readily available which is why the roles's names in DPDF are represented by a sequence of all respectively involved activities, i.e. "[Bill Account, Create Invoice, Concel Order]" and "[Notify Customer]". According to the discovered information about roles and resources in the system the respective entities and their associations are created or updated in the DBPMRT.

5.5.3 Performance

The interpretation of the performance footprint is in the case of the *activity networking time* and the *process instance occurrence* executed in a simple way: The mean value *mean* is identically taken over and the variance *var* is translated into a standard deviation $sd = \sqrt{var}$ as modelled in the DBPMRT (see Section 4.2.3).

The computation of the *path probabilities* for each decision in the control-flow on the basis of the *directly-follows* relation is done following these steps:

1. In a first step the decision $d \in S$ is traced back in the control-flow (this perspective is required input to find the path probabilities) via a breadth-first graph algorithm to find all source elements (activities or (BP) start event) $source(d) \subseteq A \cup \{\omega\}$ from which the decisions can be reached[18]. Additionally, for each outgoing path of the

[18] Sets and elements defined in BP Definition 3 on page 137

decision all elements (activities or (BP) end event) are collected, again via breadth-search, i.e. $target(d) = \{p_1, p_2, ..., p_n\}$ with $p_1, p_2, ..., p_n \subseteq A \cup \{\epsilon\}$.

2. In a second step, each path's p_i probability $pp(p_i)$ is calculated by adding for each of the decision d sources $x \in source(d)$ the sum of all directly-follows values ($x \gg y$) for each of the path's targets $y \in p_i$ relative to the overall occurrence of the respective source elements $Oov(x)$:

$$pp(p_i) = \sum_{x \in source(d)} \frac{\sum_{y \in p_i} (x \gg y)}{Oov(x)}.$$

3. In a final step all path probabilities $pp(p_i)$ of the decision d are normalised with $PP(d) = \sum_{p \in target(d)} pp(p)$ to enforce that their sum is 1, i.e. $ppn(p_i) = \frac{pp(p_i)}{PP(d)}$, which then reflects the real path probabilities in the system. All final normalised path probabilities $\{ppn(p_i)\}$ are then updated in the DBPMRT.

There are two policies for the performance interpretation lifecycle: (1) change the respective value whenever associated parts of the footprint are updated, or (2) whenever an interpretation is requested, i.e. simultaneously with the control-flow interpretation. In the DPDF the latter option is set as default.

5.6 Process Instance State Tracking

While the previous sections were concerned with establishing the causal connection for aggregated type level information (control-flow, resources, performance) this section is concerned with tracking the fine granular changes in the system on the instance level. Since every event basically constitutes a change on the instance level, the model transition frequency is a lot higher than changes in the type level information. Storing a fixed number, or even all of the past events, are not appropriate approaches for tracking the state in real-time: Whereas the first option is prone to failure due to the possibility of multiple states for parallel paths, the second option requires significant post-processing to determine the actual state since a lot of events are already outdated, i.e. overwritten.

In this section a highly run-time optimised method is presented that tracks the state of a given well-structured business process while processing the events and thus allows for an always up to date state representation without further post-processing. The state update operation for each event has a quasi-constant run-time due to exploiting bit-operations.

5.6.1 Additional Preliminaries

Additionally to the previous definitions from Section 5.2.1, the functions *BIT*, *AND*, and *OR* are specified. They are equivalent to bit-manipulation operations of common programming languages like Java ("<<" shifting bit string to left by a specified number of

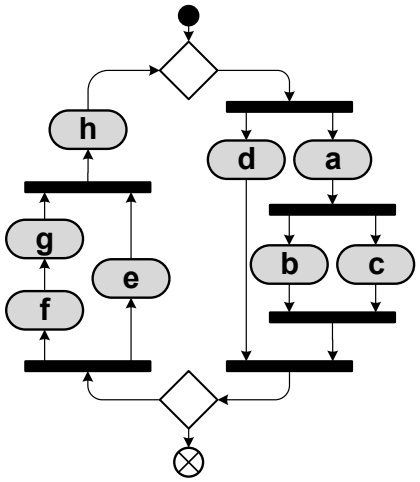


Fig. 5.13 Example Business Process with many Parallel Paths

bits, "|" bit-wise *or*, and "&" bit-wise *and*) with bs_1, bs_2 being native types that allow bit-manipulation operations (like *Integer* or *Long*) and bit index $idx \in \mathbb{N}$:

$$\begin{aligned} BIT(idx) &= 1 \;\; << \;\; idx \\ AND(bs_1, bs_2) &= bs_1 \;\; \& \;\; bs_2 \\ OR(bs_1, bs_2) &= bs_1 \;\; | \;\; bs_2 \end{aligned} \tag{5.12}$$

Furthermore, the highly parallel example process depicted in Figure 5.13 serves as reference process to explain specific details and features of the proposed approach.

5.6.2 Method Specification

The method is divided in two separate parts: (1) creating bit masks from a given well-structured BP model to pre-configure the state change behaviour for certain event occurrences and (2) a procedure which is called for every event that updates the state according to the pre-calculated bit masks. That means, preferably the BP is already known for the state tracking since it allows to determine whether a historical event is still part of the system's state or has already been made redundant. In the following it is assumed that this type level information is known. Note, that the DPDF also allows for state tracking without a given BP model through capturing and retaining all events of active traces. This is, however, a rather trivial solution and requires unnecessary intensive computational efforts for each interpretation rather than only for each type level change (like in the proposed approach).

Bit Mask Creation From BP Model:

The tracking of the BP state is essentially based on bit mask operations. For this, in a first step the BP model is translated into simple bit masks/sequences. Two different maps are created: *activityBitMap*, defining which bit is representing what activity, and

changeBitMask, which defines the activities that are in parallel paths from this activity, e.g. for activity d in the example in Figure 5.13: *activityBitMap*[d] = (...00001000) and *changeBitMask*[d] = (...00000111) since activities a, b, and c represented by bits 1-3 are parallel to d. Algorithm 3 shows the approach followed to compute these two maps: First the *activityBitMap* is filled in a straight-forward fashion (see lines 4 and 7-11). In a second step the BP model is traversed in a recursive way to find parallel paths to fill the *changeBitMask* map. Lines 13-21 and 38-41 is a simple search (breadth-first if *elementsToReview* is a queue and depth-first if it is a stack) for activities in this path on the same recursion level. If a fork element is encountered, a special routine (lines 22-37) is evoked that analyses the different outgoing paths individually through a recursive call of `traversePath` and combines the results to respective bit masks indicating which activities are in parallel to each other. These bit masks are stored in the *changeBitMask* map. The result for *activityBitMap* and *changeBitMask* of the algorithm for the BP model in Figure 5.13 is[19]:

activityBitMap:

	...	h	g	f	e	d	c	b	a
a	...	0	0	0	0	0	0	0	1
b	...	0	0	0	0	0	0	1	0
c	...	0	0	0	0	0	1	0	0
d	...	0	0	0	0	1	0	0	0
e	...	0	0	0	1	0	0	0	0
f	...	0	0	1	0	0	0	0	0
g	...	0	1	0	0	0	0	0	0
h	...	1	0	0	0	0	0	0	0

activityChangeBitMask:

	...	h	g	f	e	d	c	b	a
a	...	0	0	0	0	1	0	0	0
b	...	0	0	0	0	1	1	0	0
c	...	0	0	0	0	1	0	1	0
d	...	0	0	0	0	0	1	1	1
e	...	0	1	1	0	0	0	0	0
f	...	0	0	0	1	0	0	0	0
g	...	0	0	0	1	0	0	0	0
h	...	0	0	0	0	0	0	0	0

State Update:

After the two bit mask maps have been computed from the BP control-flow model the state can be tracked with the method shown in Algorithm 4 (*e.traceID* is the event's unique identifier for the trace/instance and *e.activity* the reference to the activity in the model). For each event this method is called and updates the respective trace state through a combination of *AND* and *OR* operations as well as the pre-computed bit masks. Assuming that for a specific instance of the example BP in Figure 5.13 the previous events occurred in the following order: $a, b, c, d, e, f, g, h, a, d$ then the current state of this trace is (...00001001) since only a and d represented by bit 1 and 4 are the currently relevant events comprising the state of the instance, i.e. none of the other previous events are required to uniquely identify the state of the process instance: If now an event representing the execution

[19] The map is depicted as matrix; on the left are the activities which are the keys to retrieve the respective bit masks and on top of the matrix are the references to show which activity is represented what bit

Algorithm 3: Creating Bit-Masks from BP Control-flow Model

Data: *CF bp*
Result: *Map activityBitMap, Map changeBitMask*

```
1  begin
2      activityBitMap ← {};
3      changeBitMask ← {};
4      buildActivityBitMap(bp.A);
5      traversePath(bp, bp.e_s);
6      return (activityBitMap, changeBitMask);
7  Function buildActivityBitMap(activities)
8      i ← 0;
9      foreach a ∈ activities do
10         activityBitMap[a] ← BIT(i);
11         i ← i + 1;
12 Function traversePath(bp, startOfPath)
13     elementsToReview ← {startOfPath};
14     elementsVisited ← {}; pathElements ← {}; endJoin ← null;
15     repeat
16         curElem ← elementsToReview.pop();
17         elementsVisited ← elementsVisited ∪ {curElem};
18         if curElem ≠ bp.e_e then
19             if curElem ∈ bp.J_∧ then endJoin ← curElem;
20             else
21                 if curElem ∈ bp.A then pathElements ← pathElements ∪ {curElem};
22                 else if curElem ∈ bp.S_∧ then
23                     paths ← {}; pathBits ← {}; corrJoin ← null; i ← 0;
24                     foreach (curElem, t) ∈ bp.C do
25                         (corrJoin, paths[i]) ← traversePath(bp, t);
26                         pathElements ← pathElements ∪ paths[i];
27                         pathBits[i] ← 0;
28                         foreach act ∈ paths[i] do
29                             pathBits[i] ← OR(pathBits[i], activityBitMap[act]);
30                         i ← i + 1;
31                     foreach (pathIdx, curPath) ∈ paths do
32                         paraPathBits ← 0;
33                         foreach paraPathIdx ∈ {0, ..., |paths| − 1} : pathIdx ≠ paraPathIdx do
34                             paraPathBits ← OR(paraPathBits, pathBits[paraPathIdx]);
35                         foreach act ∈ curPath do
36                             changeBitMask[act] ← OR(changeBitMask[act], paraPathBits);
37                     curElem ← corrJoin;
38                 foreach (curElem, t) ∈ bp.C : t ∉ elementsVisited ∧ t ∉ elementsToReview do
39                     elementsToReview ← elementsToReview ∪ {t};
40     until elementsToReview = {};
41     return (endJoin, pathElements);
```

Algorithm 4: Updating the BP State

```
Data: Map activityBitMap, Map changeBitMask, Map traceStates, Event e
Result: updated traceStates
1 begin
2   if ∄ traceStates[e.traceID] then traceStates[e.traceID] ← 0;
3   else traceStates[e.traceID] ← AND(traceStates[e.traceID], changeBitMask[e.activity]);
4   traceStates[e.traceID] ← OR(traceStates[e.traceID], activityBitMap[e.activity]);
```

of b occurs the state is updated with $OR(AND(...00001001,...00001100),...00000010) = (...00001010)$ marking activities b and d relevant for the trace's state; this again is followed by the occurrence of an event representing the execution of c which updates the trace's state to $OR(AND(...00001010,...00001010),...00000100) = (...00001110)$; if this is followed by e the state will be updated to $OR(AND(...00001110,...01100000),...00010000) = (...00010000)$ and so on.

Considering the dynamic nature of some BP's the bit masks might not (yet) account for the state of some of the events occurring. These events are ignored for the instance state and only taken into account once the information on the type level has been updated and as a result the state tracking uses the updated bit masks. For these transitional phases the state tracking might not represent an accurate reflection of reality. This behaviour can under the given circumstances not be further mitigated (there are no events for type level change, i.e. they have to be aggregated from low level events which again leads to a certain causation delay) but at the same time it only constitutes a negligible factor since the reasoning in transitional phases is relatively error prone in general due to being based on outdated information (see Chapter 6).

Algorithm Modifications for larger BP Control-flow Models

The approach above allows to process events and keep track of the BP's state in very low constant run-time provided that the model does not consist of more than 64 activities and a native type of 64 bits (like *Long*) is used[20]. However, in order to support bigger models the operations have been extended to work on arrays of native types. That means, in Algorithm 3 lines 27 and 32 assign an array of bit strings (all bits set to zero) as well as in Algorithm 4 line 2. Furthermore, the operations *BIT*, *AND*, and *OR* were adapted to similarly operate on arrays of bit strings.

5.7 Overview of Mapping from Event Lifecycle Type to BP Perspective

While in the previous section specific details about establishing and maintaining a causal link from BPMSs to a DBPMRT were discussed, this section is concerned with the general

[20] modern CPUs are based on 64-bit operations

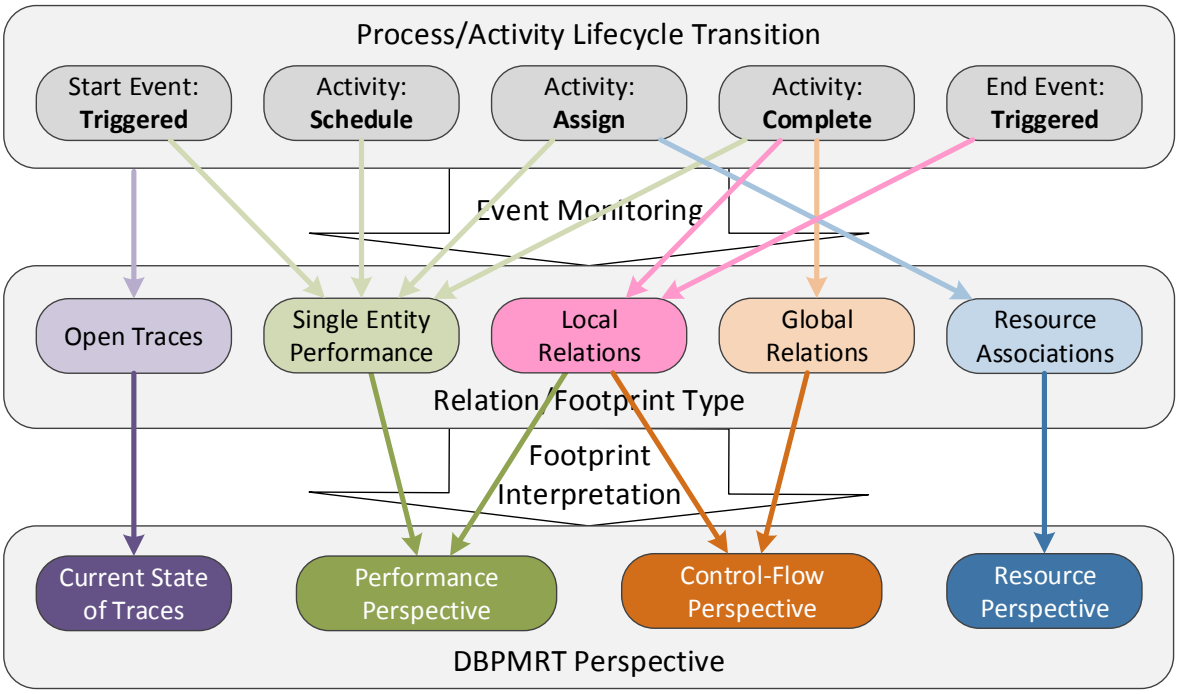


Fig. 5.14 Mappings from Event Lifecycles to Relation Types to DBPMRT Perspectives

aspect of mapping source to target information. The overview provided in Figure 5.14 shows the following three levels of data:

1. *Event Lifecycle Transition Types* as shown at the top of the figure. Five different types are of relevance for the DPDF (as also highlighted in Figure 5.8, page 154): Triggered (by a BP start event[21]), Schedule, Assign, Complete, Triggered (by a BP end event[21]).

2. *Relation Types* as shown in the middle of the picture. The five different types are not distinguished between footprints (as in the overall DPDF method in Figure 5.7, page 153) but rather between type of content: Open Traces (bit masks containing the state), Single Entity Performance (such as activity net working time and process instance occurrence), Local Relations (directly follows matrix), Global Relations (the CCM footprint as explained in Section 5.2.3), and Resource Associations (activity to resource matrix as defined in Section 5.4.2).

3. *DBPMRT Perspectives* as shown at the bottom of the figure.

With this overview the sources of the target perspectives can be tracked down to the event level. For instance, the current state of traces is calculated using the complete set of different event types. The same applies for the performance perspective albeit in a more indirect way: While activity networking time and process instance occurrence is directly calculated through Single Entity Performance information, the path probabilities are computed via the directly-follows relation (= Local Relations) - the union of both of their sources encompasses all lifecycle types.

[21] BP start events or end events are elements in the BP control-flow - not to be confused with the events emitted by a BPMS

5.8 Evaluation

This section is concerned with the evaluation of selected parts of the dynamic process discovery methodologies described in the chapter. Note, that while established methods exist for the evaluation of "offline" process control-flow discovery this is not the case for online process discovery algorithms or the discovery of holistic BP models[22]. The individual components and methodologies are evaluated in the following way: (1) Evaluation of the control-flow discovery with the CCM via established conformance measures is carried out in Section 5.8.1; (2) To evaluate the ageing methods and the behaviour of the online control-flow discovery via the DCCM a sensitivity analysis is carried out in Section 5.8.2; (3) The dynamic discovery of the holistic DBPMRT can be quantitatively evaluated through a proxy method: Comparing predicted developments of PPIs with the actual PPI values. Since this requires further specifications of higher-level reasoning it is carried out as part of the DBPMRT reasoning chapter 6. However, a qualitative evaluation of the DPDF against the requirements defined in Section 5.1 is carried out in the concluding evaluation Section 5.8.3.

5.8.1 Evaluation of Constructs Competition Miner

In this section the CCM is evaluated: First qualitatively and later quantitatively in comparison to other miners. For the tests the CCM was configured as follows: The tolerance threshold was set to $t_t = 0.001$ and the unequal penalty to $p_u = 1.0$ (see Definition 9). Furthermore, for the construct suitability computation the constraint weights were set to $w_s = 0.6$, $w_{lc} = 0.3$, and $w_i = 0.1$ (see page 148).

Rediscovery Tests

In a first evaluation step a number of initial process rediscover tests have been carried out. For this 67 example processes were created, each consisting of a small number of activities nested in a combination of BP constructs. The 67 example processes represent all reasonably sensible (from the author's perspective) nested combinations of the supported BP constructs shown in Figure 5.5 on page 147. The artificially constructed BPs were simulated to produce a corresponding log, which in turn was analysed with the CCM. The CCM rediscovered all but 4 of the conceptual processes or found a model with equivalent behaviour. The not successfully rediscovered models were variations of the Loopover-Parallel construct that had at least one loop in one of the parallel paths.

Additionally, the CCM has been tested for rediscovering more complex BPs consisting of $10-50$ activities. For this randomly created models were constructed by taking a plain sequential model and repeatedly applying random mutations on it, e.g. introduce choice, introduce loop, etc. Those strongly nested BPs were again simulated and analysed by the CCM in order to (re-)discover the initial BP model. The log and the mined model

[22] This will be evaluated through a case study in Chapter 6

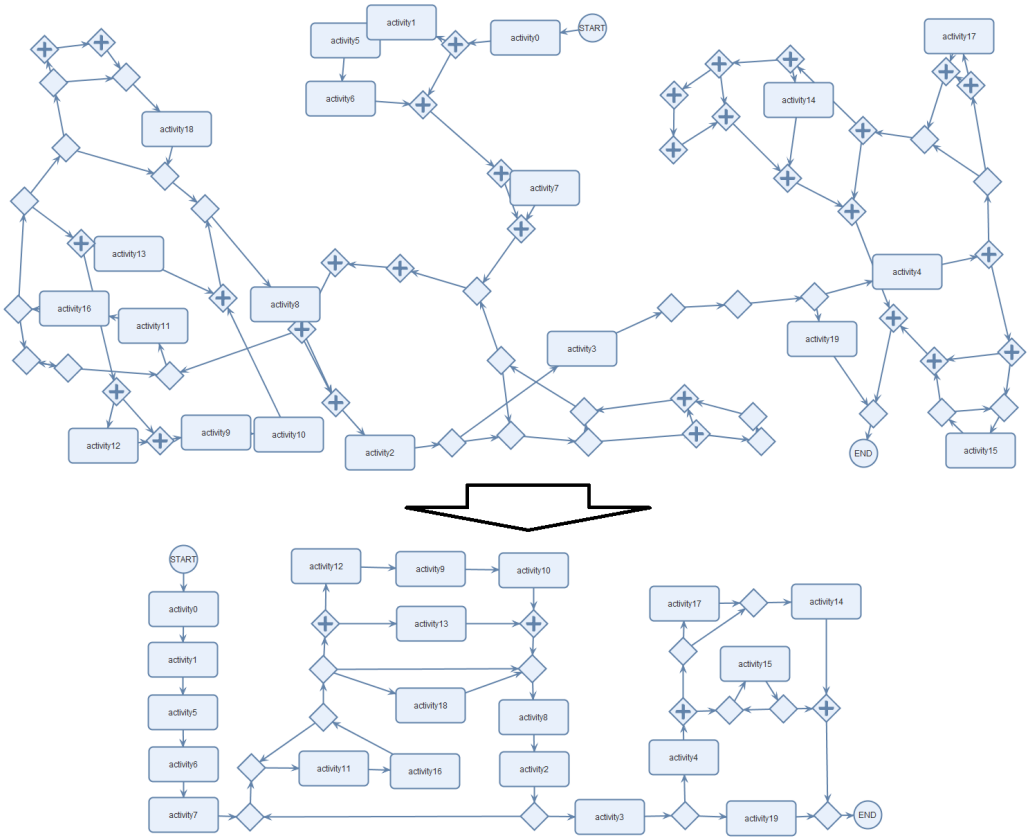


Fig. 5.15 Randomly Created Original BP (Top) and the (Re-)Discovered BP (Bottom) by analysing the simulated log of the original BP with the CCM Algorithm

were then checked for their conformance. The conformance checking method used measures the quality of the mined models in two directions: (1) *recall*, i.e. how much of the behaviour expressed by the event log is captured by the mined model, and (2) *precision*, i.e. how much of the behaviour the mined model allows for is actually supported by the log. The algorithms used for extracting and comparing log and model footprint are described in [146]. For 10 activities and 20 mutations (10 runs) an average recall of 1.0 (i.e. 100%) and a precision of 0.96 was achieved. For 20 activities and 50 mutations (12 runs of which one example is shown in Figure 5.15) an average recall of 0.97 and an average precision of 0.92 was achieved. The run with the lowest precision of 0.79 (recall 0.98) was probably very likely due to an incomplete log since the original model had an even lower precision of 0.69 (recall 1.0) which shows that not all variations have been in the log. The result of the miner is actually the desired behaviour, for the sacrifice of excluding 2% of the exceptional behaviour the precision could be increased, i.e. the mined model had a better overall conformance than the original model; For 50 activities and 100 mutations (5 runs[23]) an average recall of 0.94 and a precision of only 0.49 was achieved. However, the average precision of the source model was only 0.6 (between 0.5 and 0.76), i.e. the log was not complete. In some of the cases the CCM could even improve the precision for the cost of minimally reducing the recall, e.g. precision: 0.76 → 0.92, recall 1.0 → 0.99.

[23] 50% of the runs failed due to memory exhaustion - many nested loops produce 1000s of events per trace

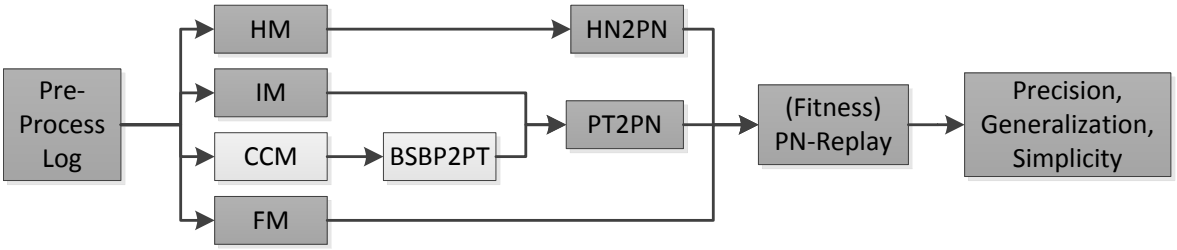


Fig. 5.16 Experimental Workflow

Comparative Evaluation: Experiment Setup

As opposed to DCCM the static CCM does not support an online causal connection from system to model but instead is an offline analysis working on an event log. It is, however, an important contribution of the work presented in this thesis and addresses functional gaps identified in Section 3.3.1: (1) it can deal with infrequent (noisy) and incomplete logs, (2) its abstraction level is that of the BP domain, and (3) it is deterministic and requires no human intervention. To the best of the authors knowledge, the only other algorithm which fulfils these requirements is the Inductive Miner (IM) including its various extensions. The CCM and IM share some concepts (e.g. the top-down-divide-and-conquer approach) but were developed entirely independently and have significant differences: (1) the IM is based on local (directly-follows) rather than global relationships (eventually-follows) and (2) it has a potentially exponential run-time when considering incomplete logs (see Section 3.3.1).

In order to quantitatively compare the performance of the CCM with state-of-the-art algorithms like the IM a conformance analysis of the discovered models was carried out with the help of the ProM framework [253]. The other investigated algorithms are the HeuristicsMiner (HM), which is a popular discovery algorithm that can deal with real-life (noisy and incomplete) logs and is thus often used as reference algorithm, and the Flower Miner (FM)[24]. The IM, HM, and FM are readily available in the nightly build of ProM and were used to benchmark the quality of the models discovered by the CCM. The experimental setup is conceptually shown in Figure 5.16: (1) The logs were filtered so that only events with the lifecycle "complete" are considered. These logs are available in the XES-format [90, 258]. (2) In a second step each individual miner discovers the process in its representation language using its default settings, i.e. HM creates a heuristic net, IM creates a process tree, CCM creates a block-structured BP, and FM directly creates a Petri net. (3) In Figure 5.16 the different transformations from the originally mined language to a Petri net representation are shown. Note, that a transformation has been implemented to translate the block-structured BP into process trees in order to enable an analysis of the CCM results with the ProM framework. (4) The Petri net representation of each mined model is then analysed with the help of the *PNetReplayer* package, an implementation of the approaches in [2, 3, 236]. With the help of this plugin, three different quality mea-

[24] Creating the Flower Model as shown in Figure 2.13 on page 36

sures are calculated that represent the conformance of the (filtered) log to the discovered model[25]: trace fitness f_{tf} - a measure how well the traces in the log can be replayed by the Petri net, precision f_{pr} - a measure how closely the behaviour in the log is represented by the Petri net, and generalisation f_g - a measure that shows to what level a generalisation of the log behaviour was achieved. Additionally, all places, transitions, and arcs of the discovered Petri nets are counted and accumulated to a simplicity measure f_s. These conformance measures are discussed in more detail in Section 2.4.1. The following 10 logs have been used for evaluation:
(1) $\mathbf{L_1}$ (8 activities, 34 traces, 204 events): log on page 35 (BP model in Figure 2.12),
(2) **EX5** (14, 100, 1498): example log of a reviewing process[26],
(3) **REP** (8, 1104, 7733): example log of a repair process[26],
(4) **BE1** (20, 8204, 189242), (5) **BE2** (20, 8206, 132235), (6) **BE3** (20, 8194, 239318),
(7) **BE4** (20, 8153, 253784), (8) **BE5** (20, 8190, 151604): large logs of strongly nested BPs that are artificially generated by a process simulation tool (similar to the rediscovery tests),
(9) **DF** (18, 100, 3354): an incomplete real-life log of an eHealth process [74], and
(10) $\mathbf{FL_A}$ (10, 13087, 60849): a large real-life log from the finance sector[27].

Comparative Evaluation: Experiment Results

The results of the different discovery algorithms applied to the investigated logs are shown in Table 5.2 for trace fitness f_{tf} and precision f_{pr} as well as in Table 5.3 for generalisation f_g and simplicity f_s. At the bottom of the table the combined averages over all 10 logs are listed. Note, that all algorithms were executed using default settings, i.e. choosing other parameters may result in different results. In order to provide comparable results also the parameters for the CCM were fixed to the default values of $t_t = 0.001$ and $p_u = 1.0$ for all runs.

Compared with HM and IM the CCM scores a generally high trace fitness for the considered logs (always higher than 0.95) with the exception of BE5 for which a trace fitness of only 0.822 was determined. The precision values of the models discovered by CCM are mostly between the respective values scored by HM and IM but always above 0.5 - as expected FM always scores the lowest for precision. Positive exceptions are L_1, BE5, and DF for which CCM scores the highest precision. In terms of the generalisation measure, the CCM scores are average in comparison to HM and IM: yielding a high result for log L_1 but low results for DF and FL_A. Very positive results are achieved if the simplicity measure is considered: the CCM mostly discovers a model consisting of the lowest number of elements (excluding FM): far smaller than the HM models and slightly smaller than the IM models. Generally, it seems that the CCM tends to favour trace fitness, generalisation, and simplicity for the cost of a lower precision.

[25] using "Prefixed based A* Cost-based fitness" algorithm with maximum explored states = 200000
[26] *exercise5* (EX5) and *repairExample* (REP) are example logs from the ProM website (processmining.org)
[27] log from the BPI Challenge of 2012 (http://www.win.tue.nl/bpi/2012/challenge) filtered for events that start with "A", e.g. "A_APPROVED", "A_DECLINED", etc.

Table 5.2 Trace Fitness and Precision conformance results of the discovery algorithms

Log	Trace Fitness f_{tf}				Precision f_{pr}			
	HM	IM	FM	CCM	HM	IM	FM	CCM
L_1	0.679	0.863	1.0	1.0	0.532	0.529	0.224	0.550
EX5	0.985	0.935	1.0	1.0	0.495	0.560	0.120	0.529
REP	1.0	1.0	1.0	1.0	0.905	0.955	0.209	0.955
BE1	0.991	1.0	1.0	1.0	0.838	0.814	0.081	0.818
BE2	0.924	0.981	1.0	0.998	0.737	0.594	0.087	0.621
BE3	0.822	0.983	1.0	0.999	0.891	0.443	0.073	0.525
BE4	0.876	1.0	1.0	1.0	0.707	0.406	0.067	0.608
BE5	0.942	0.991	1.0	0.822	0.590	0.668	0.089	0.711
DF	1.0	0.911	1.0	0.970	0.563	0.559	0.060	0.588
FL_A	0.974	1.0	1.0	1.0	0.920	0.695	0.227	0.727
ϕ	0.919	0.966	1.0	0.979	0.718	0.622	0.124	0.663

Table 5.3 Generalisation and Simplicity results of the discovery algorithms

Log	Generalisation f_g				Simplicity f_s			
	HM	IM	FM	CCM	HM	IM	FM	CCM
L_1	0.638	0.422	0.949	0.654	86	91	33	81
EX5	0.931	0.996	0.999	0.998	155	102	51	80
REP	0.998	0.999	0.999	0.999	72	46	33	49
BE1	0.999	0.999	1.0	0.999	192	132	69	122
BE2	1.0	1.0	1.0	1.0	196	156	69	146
BE3	1.0	1.0	1.0	1.0	178	149	69	139
BE4	1.0	1.0	1.0	1.0	193	173	69	149
BE5	1.0	1.0	1.0	1.0	206	181	69	167
DF	0.914	0.906	0.982	0.832	177	136	63	121
FL_A	0.925	0.825	0.988	0.818	98	62	39	65
ϕ	0.941	0.915	0.992	0.930	155.3	122.8	56.4	111.9

Process discovery is a multi-goal optimization and it is sometimes hard to tell how well a discovery algorithm performs overall (see Section 2.4.1: balance between overfitting and underfitting). From the four resulting quality criteria, usually no aggregated metric is computed since the individual requirements differ with each use case. To put the overall performance result into a more readable format and make it more comparable Table 5.4 shows the average score of the CCM against the two relevant other algorithms HM and IM (FM is only a base line case for underfitting). Additionally, the winner in each of the individual quality criteria is coloured in green, the second place in olive green, and the last

Table 5.4 All averaged results of the relevant discovery algorithms

Trace Fitness f_{tf}			Precision f_{pr}			Generalisation f_g			Simplicity f_s		
HM	IM	CCM	HM	IM	CCM	HM	IM	CCM	HM	IM	CCM
0.919	0.966	0.979	0.718	0.622	0.663	0.941	0.915	0.930	155.3	122.8	111.9

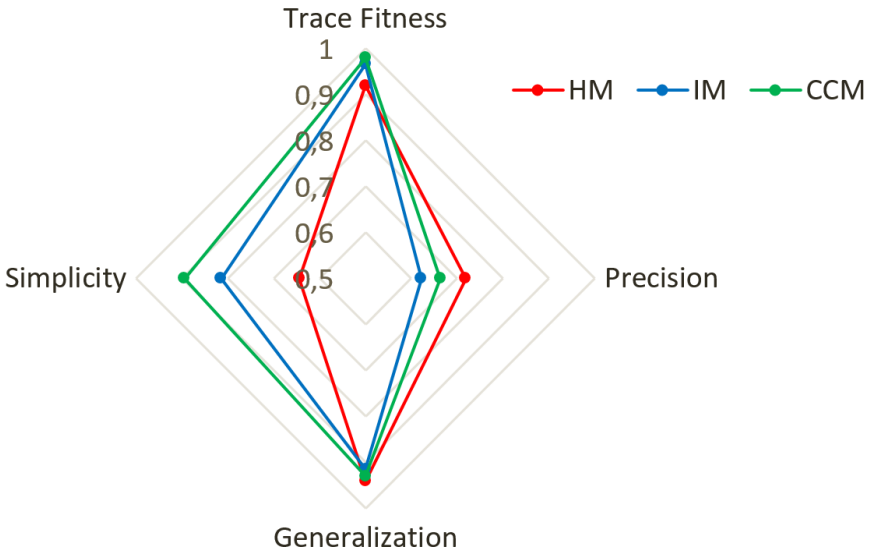


Fig. 5.17 Quality Criteria Net of HeuristicsMiner (HM), Inductive Miner (IM), and Constructs Competition Miner (CCM); Simplicity values normalised with $|f_s| = \frac{100}{f_s}$

place in brown. The CCM scores best in trace fitness and simplicity and second best in precision and generalisation. The HM scores best in precision and generalisation but as a trade off very badly in trace fitness and simplicity. While the IM has more balanced results it only comes second in trace fitness and simplicity and last in precision and generalisation. To get a better picture of the overall performance Figure 5.17 shows the average score of the individual discovery algorithms as net graph. In order to fit the simplicity score into the same range ($0.5 \leq f \leq 1.0$) and direction (higher is better) the absolute simplicity score of each discovery algorithm has been normalised to $|f_s| = \frac{100}{f_s}$, i.e. HM: $|f_s| = \frac{100}{155.3} = 0.644$, IM: $|f_s| = \frac{100}{122.8} = 0.814$, CCM: $|f_s| = \frac{100}{111.9} = 0.894$. The picture shows that on balance the CCM performs better than HM and IM since it scores either the highest or close to the highest in all relevant criteria.

5.8.2 Evaluation of Dynamic Constructs Competition Miner

In this section evaluation results of the DCCM are presented with regards to its capability of initially discovering the correct process and how it reacts to certain changes of a real-time monitored business process. Furthermore, a comparative analysis is carried out to determine the advantages and disadvantages of the two ageing strategies, discrete and time-dependent ageing.

Experiment Setup

The experiments revolve around the concept of process rediscovery, i.e. that a source business process is executed and produces an event stream which is fed to the DCCM which then should discover a process behaviourally equivalent to the executed source process. Figure 5.18 shows three measures (t_w, t_d, and t_{tr}) which are used to evaluate the

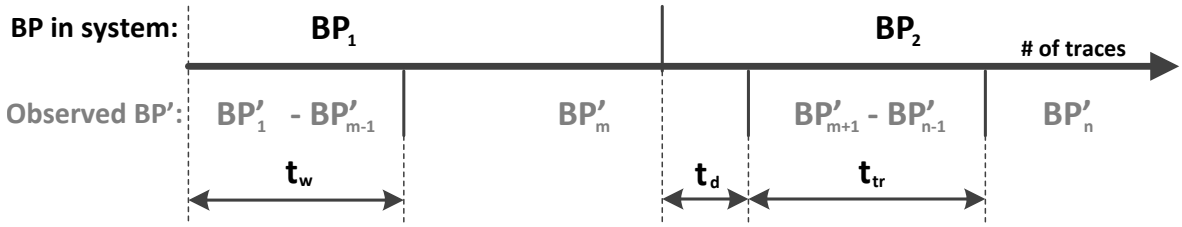


Fig. 5.18 Measures for Detection of BP Change in System

quality of the DCCM. In the figure BP_1 and BP_2 are the business processes deployed in the monitored system and BP'_1 to BP'_n are the (control-flow) models discovered by the DCCM. Additionally, BP_1 and BP'_m are equivalent ($BP_1 \equiv BP'_m$) as well as BP_2 and BP'_n ($BP_2 \equiv BP'_n$). For this part of the evaluation the following measures are of interest:

- Warm-up: $t_w \in \mathbb{N}$ the amount of completed traces the DCCM needs as input at the start until the resulting model equivalently represents the process in the system, i.e. until $BP_1 \equiv BP'_m$.
- Change Detection: $t_d \in \mathbb{N}$ the amount of completed traces it takes to detect a certain change in the monitored process - from the point at which the process changed in the system to the point at which a different process was detected. When the change is detected the newly discovered process is usually not equivalent to the new process in the system BP_2 but instead represents parts of the behaviour of both processes, BP_1 and BP_2.
- Change Transition Period: $t_{tr} \in \mathbb{N}$ the amount of completed traces it takes to re-detect a changed process - from the point at which the process change was detected to the point at which the correct process representation was identified, i.e. until $BP_2 \equiv BP'_n$. In this period multiple different business processes may be detected, each best representing the dynamic footprint at the respective point in time.

The basis of this evaluation is the example model in Figure 5.13 on page 174 which is simulated and the resulting event stream fed into the DCCM. Since the simulation is non-deterministic this is repeated 60 times for each configuration in order to get reliable values for the three measures t_w, t_d, and t_{tr}. From these 60 runs the highest and lowest 5 values for each measure are discarded and the average is calculated over the remaining 50 values. For each experiment the CCM core is configured with its default parameters (see beginning of Section 5.8.1).

Warm-up Evaluation

The first experiment will evaluate how the DCCM behaves at the beginning when first exposed to the event stream, more particularly, the duration of the warm-up phase t_w is determined. Figure 5.19 shows the development of the first few BP models extracted by the DCCM using *Discrete Ageing* with *trace influence factor* $t_{if} = 0.01$ (see Section 5.4.1)

and *interpretation frequency* $m = 10$, i.e. an interpretation is executed every 10 completed traces: After the first trace the discovered process is a sequence reflecting the single trace that defines the process at that point in time. At trace 10, which is the next scheduled footprint interpretation, the algorithm discovers a Loop construct but cannot further analyse the subsets since the corresponding sub-footprint was not requested yet. Because of that, the feedback mechanism via the *Sub-Footprint Configurations* is utilised by the *Footprint Interpretation* algorithm to register the creation of the missing sub-footprints. In the next scheduled run of the footprint interpretation, the Parallel construct of a, b, c, and d is discovered but again the analysis can not advance since a sub-footprint for the individual activity subsets has not been created yet. Activities e, f, g, and h seem to have appeared only in exactly this sequence until trace 20. Skipping one of the interpretation steps, it can be seen that at trace 40 the complete process has been mined, i.e. $t_w = 40$.

In Figure 5.20 the development of warm-up duration t_w for of the DCCM with *Discrete Ageing* for different $m \in \{1,2,3,6,10\}$ and $t_{if} \in \{0.001, 0.002, 0.005, 0.01, 0.018, 0.03\}$ is depicted. The warm-up phase seems generally very short and not strongly influenced by t_{if}. For $m = 10$ the warm-up phase cannot be any shorter because the example process consists of a block-depth of 3: Parallel-in-Parallel-in-Loop, i.e. 3 subsequent requests for sub-footprints have to be made. This is an indicator that the modification effort to shorten the warm-up phase had a positive effect. No significant changes of t_w can be noticed when increasing or decreasing the *trace influence factor* t_{if}. This is caused by the optimisation rule that essentially nullifies the default ageing via t_{if} (see Equation 5.9 on page 164).

A similar sensitivity experiment was carried out for the warm-up duration t_w with the *Time-dependent Ageing*: In order to make the results comparable the *ageing time unit* was set to one minute ($t_{ur} = 1 min$) and the simulation produced on average approximately one instance per minute (instance occurrence: $1/min$; deviation: 0.5). As a result similar

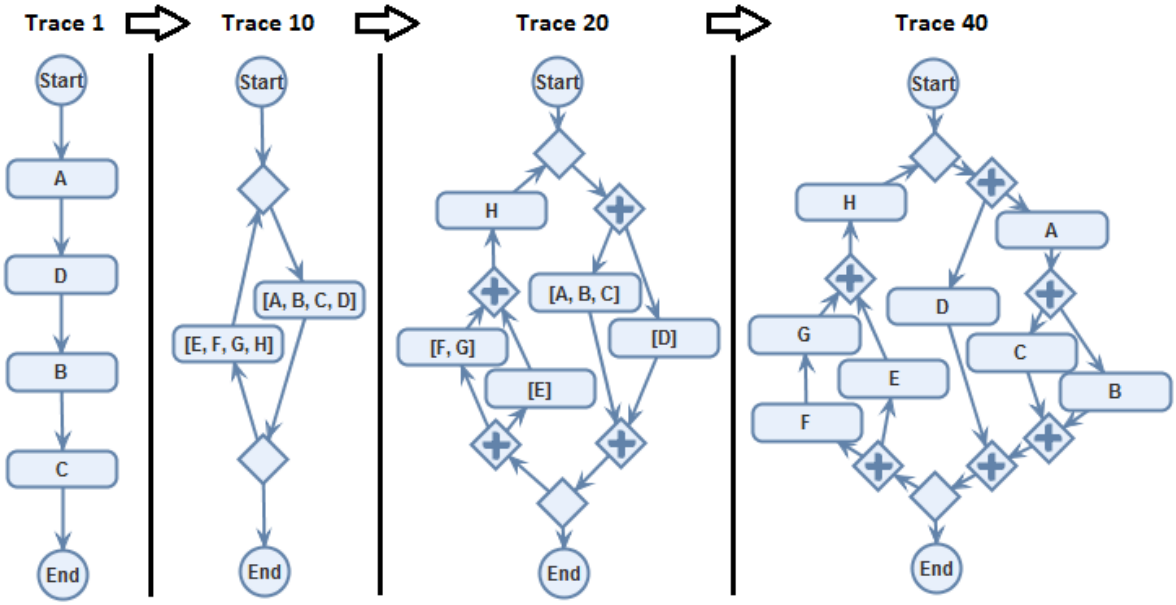


Fig. 5.19 The Evolution of the Discovered BP Model During the Warm-up Phase

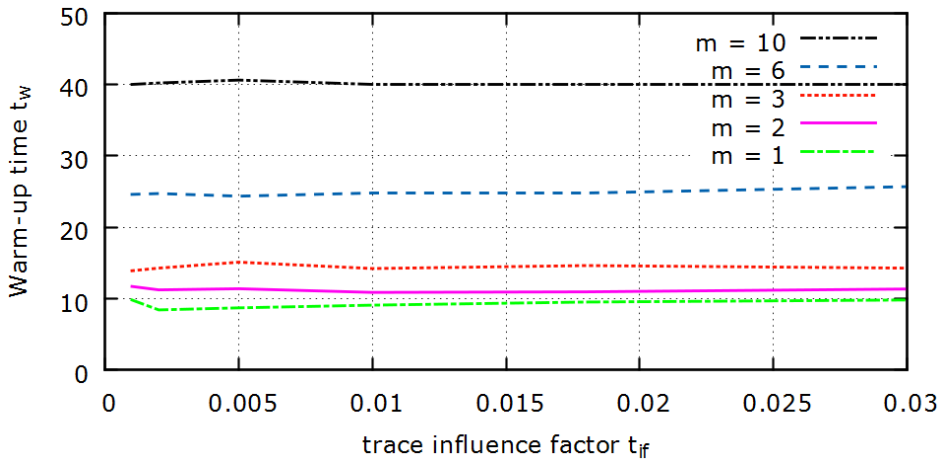


Fig. 5.20 Warm-up Time with the Discrete Ageing in Relation to the Trace Influence Factor

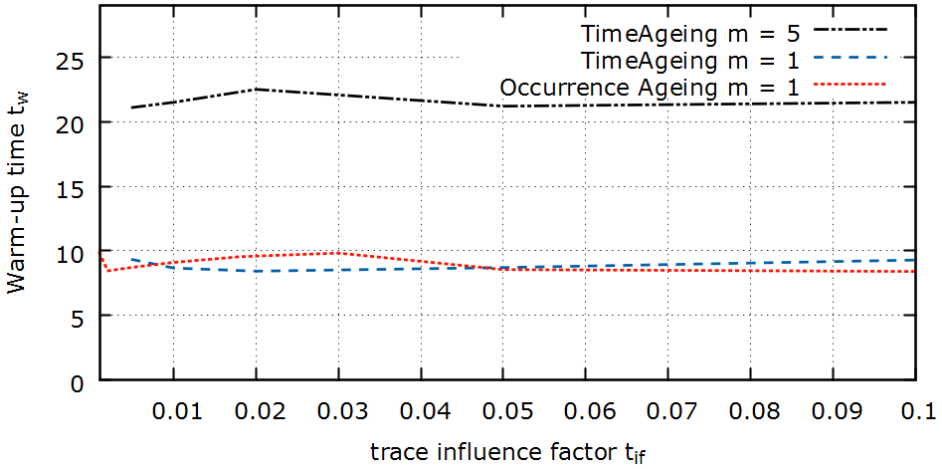


Fig. 5.21 Warm-up Time with Time-dependent Ageing in Relation to the Trace Influence Factor

t_{if} for the discrete and time-dependent[28] ageing should yield results of a similar magnitude and are thus comparable, i.e. 10 traces ≈ 10 minutes. In Figure 5.21 the development of warm-up duration t_w for the DCCM with *Time-dependent Ageing* for different $m \in \{1,5\}$ and $t_{if} \in \{0.005, 0.01, 0.02, 0.05, 0.1\}$ is depicted. As a reference the results of the *Discrete Ageing* with $m = 1$ were also added to the graph. Similar to the discrete ageing, the development of the warm-up duration t_w seems to be in no relation to the trace influence factor t_{if} for the time-dependent ageing. This indicates that the optimisation rule for the time-dependent ageing successfully improves and nullifies the default ageing method for the warm-up phase (see Equations 5.11 on page 165). Additionally, it can be seen that the result of the two different ageing types with a similar configuration yield similar results (for $m = 1$).

In order to examine the effect of the optimisations for both of the ageing methods, the same experiments were repeated without the respective optimisations (only for $m =$

[28] Time-dependent ageing is technically based on an *ageing rate* a_r rather than *trace influence factor* t_{if}. However, a_r can be derived from t_{if}, i.e. $a_r = 1 - t_{if}$

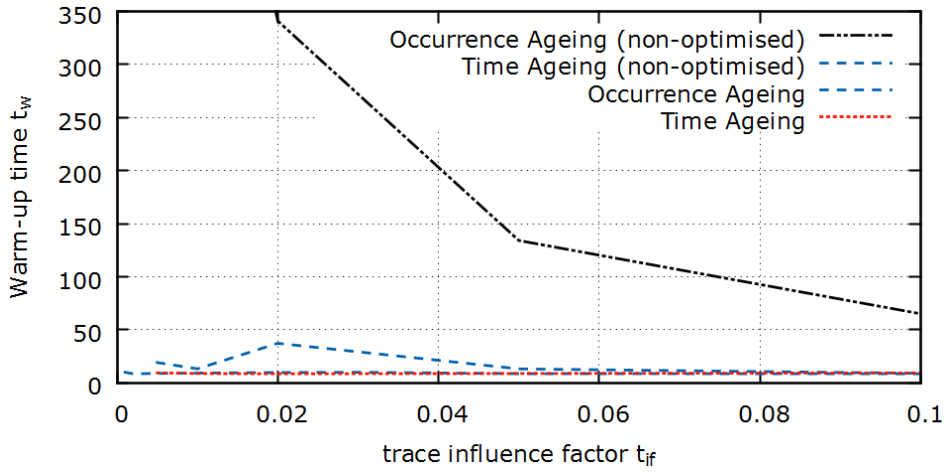


Fig. 5.22 Warm-up Time without Optimisation in Relation to the Trace Influence Factor with $m = 1$

1). Figure 5.22 shows that especially for the discrete ageing a sizeable improvement was achieved through the application of the proposed optimisations, e.g. for $t_{if} = 0.02$ the non-optimised discrete ageing took on average 341 traces until the model was rediscovered but the optimised version was already successful after 8.44 traces (on average). The non-optimised time-dependent ageing already yielded comparatively good results which could be further improved by the optimisation proposed, e.g. for $t_{if} = 0.02$ the non-optimised time-dependent ageing took on average 37.16 minutes ($\approx$ number of traces) until the model was rediscovered but the optimised version was already successful after an average of 8.42 minutes.

Change Evaluation

In a second set of experiments three changes of different extent are applied to the business process (moving of an activity, adding an activity, and a complete process swap) during execution. Of interest for these experiments are the behaviour of the DCCM as well as the change detection time t_d and the change transition period t_{tr}.

Moving of Activity "A" The change applied is the move of activity A from the position before the inner Parallel construct to the position behind it. Figure 5.23 shows the evolution of the discovered BP models with discrete ageing and *trace influence factor* $t_{if} = 0.01$ and *interpretation frequency* $m = 10$. The change was applied after 5753 traces. The footprint interpretation detects a concept drift at the first chance to discover the change (trace 5760) and finds via competition the best fitting construct: Parallel of a, c and b, d. The change detection t_d seemed to be unrelated to m and t_{if} for all experiment runs and was immediately recognised every time[29]. In Figure 5.24 the development of t_{tr} for both ageing approaches with different $m \in \{1, 10\}$ and $t_{if} \in \{0.005, 0.01, 0.02, 0.05, 0.1\}$ is

[29]Note, that other changes like deletion of an activity will take longer to recognise, since their existence still "lingers" in the footprints "memory" for some time.

shown. For this change a clear difference between the performance of the discrete and the time-dependent ageing can be observed: The time-dependent ageing is significantly slower to finally detect the correct changed process, e.g. for $t_{if} = 0.02$ and $m = 1$ the time-dependent method detects the correct process BP_2 after on average 345 traces/minutes while the discrete ageing method recognises the new process correctly already after 105 traces (on average). Furthermore, it is observable that the change transition period t_{tr} was not particularly influenced by the interpretation frequency m but strongly influenced by t_{if}. If the value was very small ($t_{if} = 0.005$) a change took on average 450 traces (discrete) or 1382 traces/minutes (time-dependent) in order to be reflected correctly in the discovered BP model. On the other hand if the trace influence factor is chosen very high, e.g. $t_{if} = 0.1$, the new process is correctly discovered after 21.1 (discrete) or 67 (time-dependent) traces/minutes.

Adding of Activity "I" The change applied in this scenario is the addition of activity I at the end of the process. Figure 5.25 shows the evolution of the discovered BP models with discrete ageing and *trace influence factor* $t_{if} = 0.01$ and *interpretation frequency* $m = 10$ while the change was applied after 5753 traces. It can be observed that the change detection t_d was again immediate, i.e. unrelated to m and t_{if}. However, it took on average 140 traces longer for the transition phase to be completed than for the previous scenario, i.e. on average 690 traces were necessary. The intermediate model which was valid from trace 5760 until 6450 (exclusively) recognises the relative position of activity I correctly (at the end of the process) but makes the activity optional. This is due to the fact that the "memory" of the dynamic overall footprint still contains behaviour from the original process in which the process ended without an activity I. Only after a certain amount of

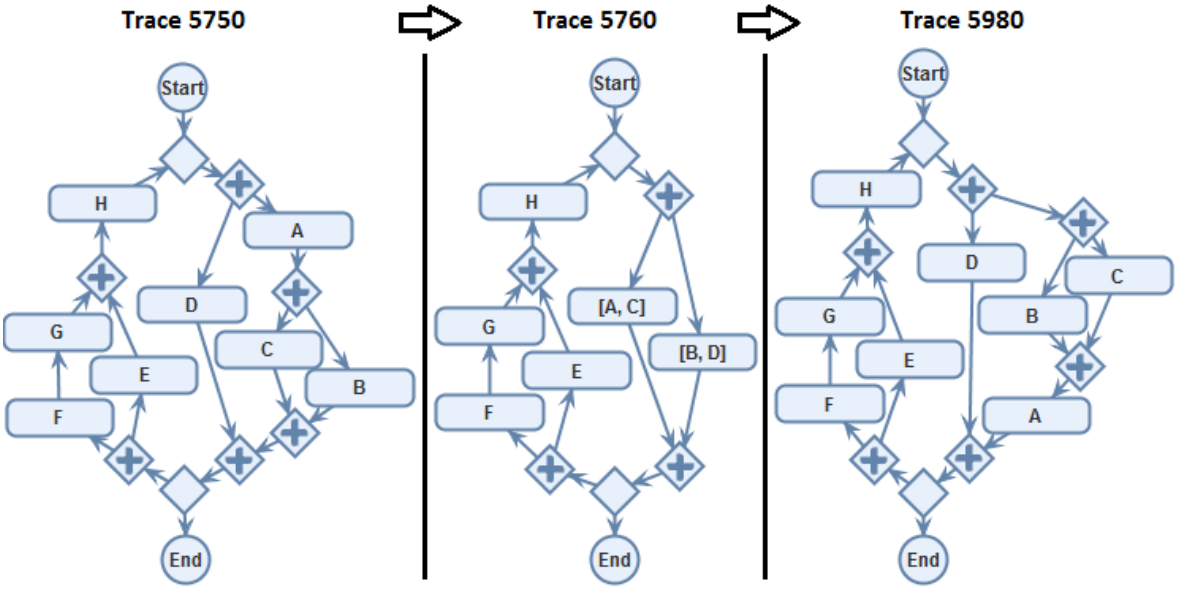


Fig. 5.23 The Evolution of the Discovered BP Model During a Change (Move of Activity "A")

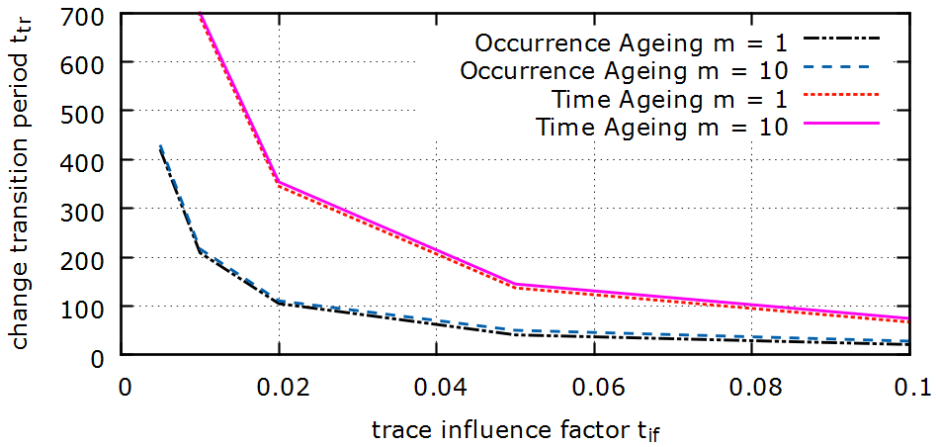


Fig. 5.24 The Change Transition Period in Relation to the Trace Influence Factor (for Recognising Move of Activity "A")

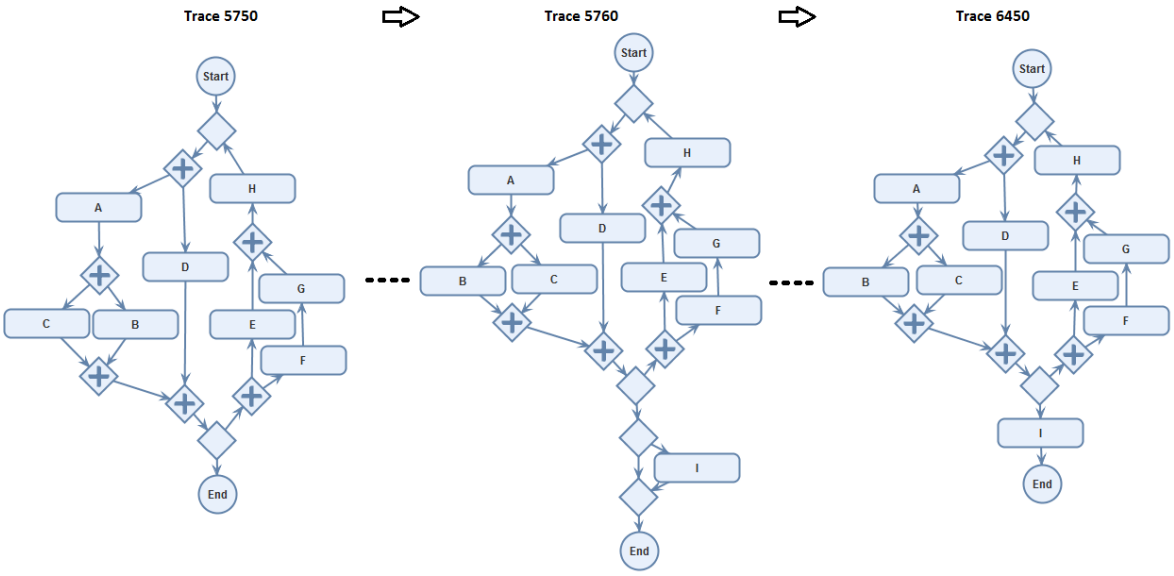


Fig. 5.25 The Evolution of the Discovered BP Model During a Change (Addition of Activity "I")

traces (dependent on the trace influence factor t_{if}) the memory of this behaviour became insignificant. Figure 5.26 shows an overview of the development of t_{tr} for all changes and both ageing approaches with $m = 1$ and $t_{if} \in \{0.005, 0.01, 0.02, 0.05, 0.1\}$. Both ageing approaches behave similarly for the change of introducing the new activity I: even though the time-dependent ageing was usually slightly quicker, e.g. for $t_{if} = 0.01$: 687 (discrete) vs. 684 (time-dependent) traces/minutes, the transition periods essentially only differed marginally (2 to 6 traces). When comparing the transition period t_{tr} for the discrete ageing method it can be concluded that the movement of an activity was correctly detected quicker than the addition of a new activity ($t_{if} = 0.01$: 209 (move A) vs. 687 (add I)). However, for the time-dependent ageing only a relatively small difference was observed ($t_{if} = 0.01$: 691 (move A) vs. 684 (add I)).

Complex Change (Swap of Complete Process) The change applied in third scenario is a complete exchange of the original process during runtime, i.e. a revolutionary change. Figure 5.27 shows the evolution of the discovered BP models with discrete ageing and *trace influence factor* $t_{if} = 0.01$ and *interpretation frequency* $m = 10$ with the change being applied after 5753 traces. The change detection t_d was again immediate (not displayed in Figure 5.27), thus being unrelated to m and t_{if}. Many different process versions occurred during the transition phase of which only the version at trace 6310 is exemplary shown in the figure. Process BP_2 which is extremely different from BP_1 was finally correctly detected at trace 6680, i.e. on average 927 traces after the change was applied and 230 traces later than the introduction of a new activity (second scenario). When comparing the performance of discrete and time-dependent ageing it can be observed that both develop similarly in relation to the trace influence factor t_{if} (see top two graphs in Figure 5.26). This scenario can be considered the baseline scenario for how long the transition phase may last at maximum for any t_{if}.

Discrete vs. Time-dependent Ageing

A first observation is that the change detection t_d for all changes and ageing configuration was always quasi-immediate, i.e. whenever the first interpretation occurred after a change it was detected that the process has changed. This is mainly due to the configuration of the core CCM component and may change if different interpretation thresholds are selected. The ageing strategy, however, seems to not influence this directly (unless an extremely low trace influence t_{if} is selected).

A second observation is that there is a significant difference for the transition duration t_{tr} depending on the extent of the change, e.g. the movement of activity A took on average only $t_{tr} = 209$ traces to be correctly recognised whereas swapping process BP_1 with an entirely different process took on average $t_{tr} = 932$ to be recognised by the DCCM with

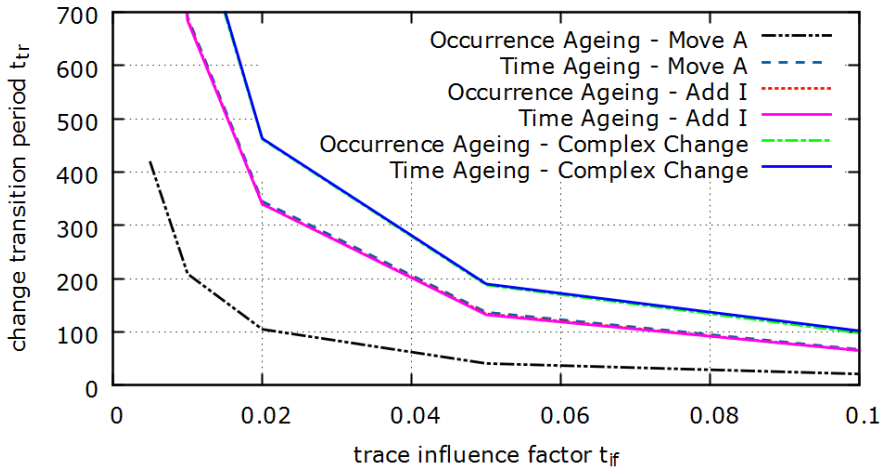


Fig. 5.26 The Change Transition Period in Relation to the Trace Influence Factor for all three Changes (m=1)

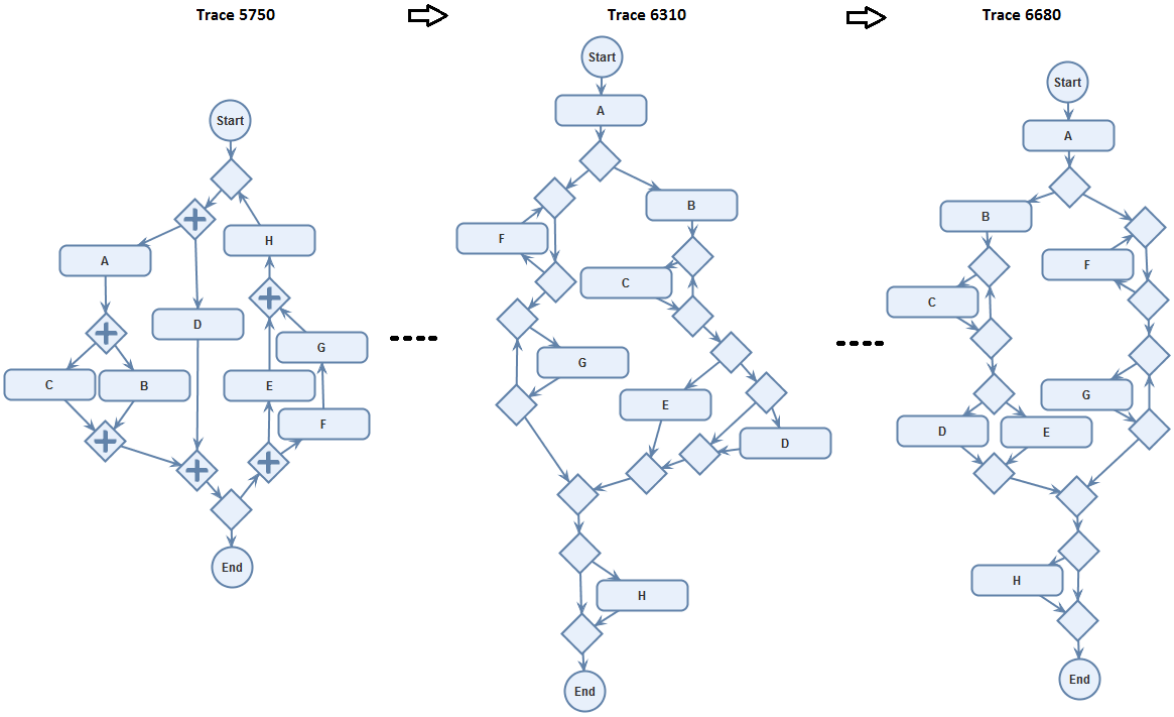


Fig. 5.27 The Evolution of the Discovered BP Model During a Change (Complex Change)

discrete ageing and $t_{if} = 0.01$ and $m = 1$.

Thirdly, time-dependent and discrete ageing perform in most cases similarly well, with the exception of change scenario 1, in which activity A was moved to a different position within the process. Here, the discrete ageing was able to detect the change significantly earlier than the time-dependent method, e.g. for $t_{if} = 0.1$ the new process is correctly discovered after 21.1 (discrete) or 67 (time-dependent) traces/minutes. However, a fact not shown in the graphs is the number of exceptional results: While the time-dependent ageing did not suffer any exceptional experiment results it was observed that in approximately 4% of the experiments for the discrete ageing method disproportionately high values for t_{tr} occurred. Due to the experiment setup, these results were ignored (i.e. removal of the five highest and lowest values). In this context it was furthermore observed that a very high trace influence $t_{if} > 0.1$ (not part of the above experiments) resulted in frequently changing/alternating discovered BP models even though the source process producing the events did not change. The reason for this behaviour is that not all variations of the process are included in the current dynamic footprint because they have already been "forgotten" before they reappeared. It was observed that for the examined scenarios the discrete ageing is slightly more susceptible to this issue than the time-dependent approach. Generally, the issue of "forgetting" too quickly is more likely to occur in large business processes containing rarely executed but still relevant behaviour and emphasises the importance of setting the trace influence factor t_{if} correctly to balance between timely correct discovery (higher t_{if}) and a sufficiently long memory (lower t_{if}) to not forget frequently occurring behaviour.

5.8.3 Qualitative Evaluation

In the final evaluation part the proposed concept and the involved algorithms are qualitatively analysed with regards to the requirements established in Section 5.1:

TR1 *Independent/Autonomous:*
The overall concept does not rely on any external output information apart from specific configuration parameters, e.g. trace influence factor t_{if}, which might have to be adapted depending on the use case. While the overall concept principally follows an autonomous approach and also promotes the independence of individual components, some of the them rely in their implementation on input from other components due to considerations with regards to improvement of the algorithmic run-time. For instance, (1) the instance state tracking preferably uses a given control-flow model as input, or (2) the directly-follows matrix is only maintained once to avoid duplicated computation effort despite being technically part of both, the control-flow footprint and the performance footprint.

TR2 *Well-fitting Result:*
As shown in the previous evaluation sections, the CCM (and the DCCM by extension) performs accuracy-wise on par or even slightly more balanced than other state-of-the-art discovery algorithms even if the events contain sporadic, contradictory, or incomplete behaviour. It has also been shown that the CCM does have problems detecting the rare and complex construct of "loop over parallel paths"[30].

TR3 *Robustness:*
The CCM always yields a control-flow output for a given log. When translated to a dynamic real-time environment (DCCM) it can be argued that this is not entirely true any more since the interpretation does not drill down further if sub-footprint configurations have previously not been registered and recorded with the control-flow footprint update component. As a result abstracted group-activities encapsulating singular-activities become temporarily part of the control-flow model, e.g. see Figure 5.10 on page 159. This results in different ramifications for other perspectives: While the resource perspective is still interpreted, performance and state-tracking is suspended until a complete control-flow interpretation (with all sub-footprints available) is carried out. Although only temporarily, this requires the DBPMRT reasoning to operate on a slightly outdated (but complete) model until this situation has been resolved.

TR4 *Detection of Change:*
Change on the instance level (reflectivity) as well as on the type level (dynamism) is detected by the DPDF as shown in the evaluations (type level in Section 5.8.2; instance level in the later cases study Section 6.3). However, changes in the system

[30] Note, that to the best of the authors knowledge other deterministic process discovery algorithms perform in that case not better or even worse.

are often not immediately and correctly reflected by the DBPMRT due to: (1) The employment of an interpretation rate m as part of the separation of FP update and interpretation, and (2) the employment of a trace influence factor t_{if} which regulates the memory and influence of new process instances. While the first was shown to have a negligible effect on the speed of change detection (see Section 5.8.2) or is even not applicable for some perspectives, e.g. resource and instance state, the trace influence factor plays a significantly more influential role for the delayed change detection. In fact, a change is immediately recognisable in almost all investigated cases. However, only after an unsatisfyingly long transition phase does the model reflect the reality (because the footprint does still contain the behaviour of the older models for some time). This effect weakens the causal link for type level information (delayed reflection of the system in the model) but is essentially due to the problem of balancing between exceptional and new consistent behaviour. This problem is difficult to mitigate if no higher-level events are available (e.g. when and how a BP has changed).

TR5 *Optimised Algorithmic Run-time/Memory Usage:*
Much effort has been carried out to optimise the algorithmic run-time and memory usage: (1) Firstly, the conceptual separation into two lifecycles (when necessary) enabled an event processing/footprint update of quasi-constant algorithmic run-time with limited memory usage (when assuming the amount of different activities/resources is not significantly increased during run-time); (2) Secondly, further run-time improvements of individual components were detailed in the respective sections, e.g. using bit operations for state tracking. However, from an end-to-end perspective (BPMS events to DBPMRT, i.e. footprint update and interpretation) the proposed solution is not particularly scalable which can be attributed to characteristics of the problem: The big gap in abstraction and the lack of higher-level events emitted by common BPMSs.

TR6 *Extensibility:*
Extensibility is to a high degree supported by the main concept since it follows a modular structure: new adapters for different event formats can be added and processing components can be exchanged/removed/added depending on the purpose and extent of the DBPMRT that is to be extracted. However, as pointed out in the qualitative evaluation of TR1, a number of components have been implemented using interfaces to exchange information across the different perspectives in order to optimise the algorithmic run-time. This slightly increases the requirements for components (since they also have to implement the interfaces).

The degree of fulfilment as semi-quantitative measure for each of the requirements based on the results of the qualitative analysis is shown in Table 5.5.

Note, that the DPDF is quantitatively evaluated for its capability to discover a holistic DBPMRT for BP simulation in Chapter 6.

Table 5.5 Degree of Requirement Fulfilment

(++ = Full, + = Mainly, ∘ = Partly, − = Negligible)

Requirement:	Principal Fulfilment	Restrictions	**Degree of Fulfilment**
TR1	Yes	- single components not independent from components of other perspectives - configuration parameters	+
TR2	Yes	- CCM has problems detecting loop over parallel paths (rare)	++
TR3	Yes	- creates group-activities in transition phase (hinders reasoning)	+
TR4	Yes	- long transition phase	+
TR5	Yes/No	- Yes: only for footprint update - No: if interpretation (in separate lifecycle) is considered as well	∘
TR6	Yes	- due to interconnection of components more interfaces need to be impl. by new component	+

5.9 Summary

In this chapter the establishment of a causal link from BPMSs to the DBPMRT was discussed in four parts: First, the problem was further specified and requirements to address it were formulated in Section 5.1. Then in the second part (Section 5.2) the Construct Competition Miner (CCM) is presented: a novel algorithm that discovers the BP control-flow from an event log potentially containing exceptional and/or contradictory behaviour. The CCM follows a divide and conquer approach to directly discover block-structured control-flow models which consist of common BP-domain constructs and represent the main behaviour of the process. For each recursion step of the divide and conquer methodology the different supported constructs compete with each other for the most suitable solution from top to bottom using constraints and behaviour approximations based on global relations between activities. The CCM represents the foundation for establishing a causal link from BPMS to the control-flow perspective of the DBPMRT. In the third part the general concept, involved agents, and specific aspects for *maintaining* the causal link for a dynamic run-time environment, the Dynamic Process Discovery Framework (DPDF), were discussed (Sections 5.3-5.7). For this the CCM approach has been expanded along two dimensions: (1) dynamic control-flow discovery at run-time, i.e. modifying the CCM to create the Dynamic Construct Competition Miner (DCCM), and (2) including additional perspectives into the concept such as resources, performance, and instance state. Notable contributions of the DPDF are the employed ageing concept based on time-dependent or distinct ageing, as well as the time-efficient algorithm based on bit operations that tracks the fine-grained instance state of the BPMS. In a fourth and

final part in Section 5.8, the algorithms concerned with the control-flow discovery, CCM and DCCM, were individually evaluated in a quantitative fashion and the holistic DPDF concept was qualitatively evaluated against the requirements established in Section 5.1. The qualitative evaluation against the requirements of the problem definition provided theoretical evidence that the approach of the DPDF is a first successful step towards establishing and maintaining a *causal* link from enterprise system to a descriptive business process run-time model. Practical evidence of DPDF's usefulness is provided in the next chapter for the use case of performance prediction.

Chapter 6

Reasoning on Descriptive Business Process Models at Run-time

This chapter serves a dual purpose: (1) In addressing third objective of this thesis: Reasoning on a DBPMRT, and (2) Evaluation of the usefulness of the DBPMRT reference model (Chapter 4) and the DPDF concept and algorithms (Chapter 5) with the help of a reasoning use case. The potential benefit of reasoning on a DBPMRT is that the system's complete state information can be utilised to carry out high-level analyses for decision support. Since the DBPMRT is an abstracted state representation of the system, a natural reasoning technique is that of simulation. Furthermore, adaptive reasoning in the BP domain is mostly based on expert knowledge and current or predicted Key/Process Performance Indicators (KPIs/PPIs - see Section 2.5). For this reason a relevant and promising reasoning use case is that of PPI prediction via simulation as identified in Section 3.4. If proven successful, this will not only validate the reasoning framework but by extension also give an estimate to what extent BP reasoning techniques can be improved by basing them on descriptive BP run-time models (i.e. proof of the DBPMRT + DPDF concept).

The chapter is structured as follows (see Figure 6.1): In the first part, the framework is introduced that performs the reasoning on the DBPMRT. The particular focus of this section are performance parameters and their prediction via short-term/transient BP simulation. In a second part, a retailer company as case study scenario is introduced. This includes the original specification of the company as well as key business processes and the organisational perspective. In a third part, the introduced reasoning framework (and by extension the quality of DBPMRT and DPDF) is analysed for its prediction quality with regards to the introduced scenario.

6.1 DBPMRT Reasoning Framework

This section is proposing a general overall concept for the reasoning framework (Section 6.1.1) as well as introducing important details of the proposed implementation for the use case of PPI prediction via simulation (Sections 6.1.2 & 6.1.3).

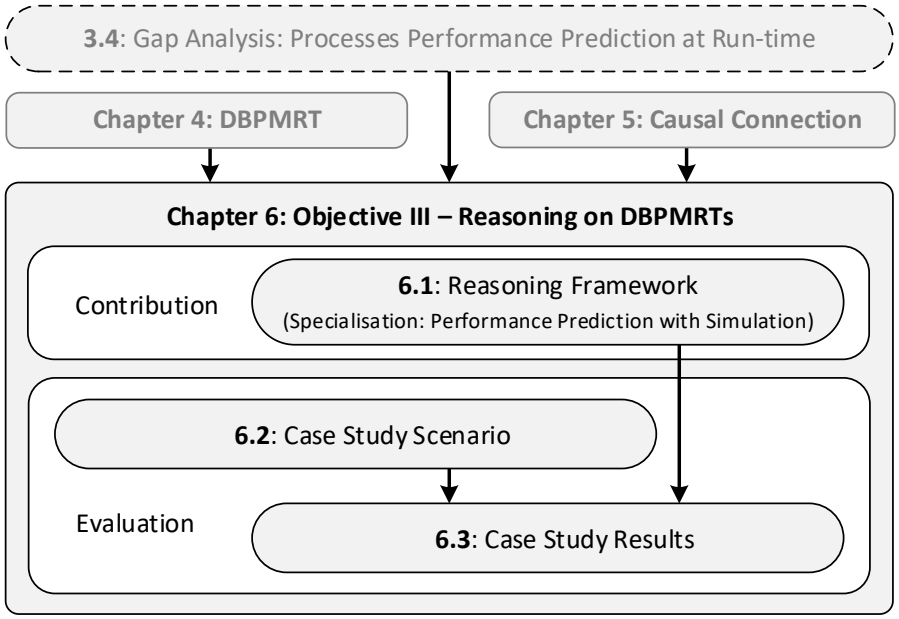


Fig. 6.1 Outline of Chapter 6

6.1.1 Concept of DBPMRT Reasoning Framework

The reasoning on DBPMRTs is in itself a models@run.time (MRT) system: The continuously executed reasoning techniques are based on a run-time model that is subject to continuous changes imposed by an outside source (the DPDF) and have to adapt accordingly. As a consequence the reasoning techniques and the DBPMRT are causally connected.

The MRT architecture adopted for the DBPMRT reasoning framework is that of a repository architecture (see Section 2.7.3). The adopted architecture is shown in Figure 6.2: The DPDF is the controlling entity, manipulating (create, add, delete, update) the repository entries, i.e. the run-time model (DBPMRT). Potentially, the repository features versioning capabilities to store older variants of the DBPMRT or, alternatively, stores a more compact Evolution DBPMRT as introduced in Section 4.3. The DBPMRT can then be accessed by various analysis/reasoning components which act as run-time systems in this architecture. There are two different types of analysis components: (1) some solely consume the DBPMRT channelling their resulting output into other components, e.g. visualisation or notifications, (2) others process information in the DBPMRT into higher-level run-time information that is part of the DBPMRT specification, i.e. model refinement (see Section 2.6.3). With the second type mega-modelled analysis chains using the DBPMRT as information hub can be supported.

Both, model update and reasoning components, have their own lifecycles and access the DBPMRT in an uncoordinated fashion. For this reason the repository is required to enforce synchronised access, e.g. the analysis component is required to (bulk-)read information from the DBPMRT associated with a specified state; during that time the state may not be updated since this could contaminate the read-out information and invalidate the analysis' results.

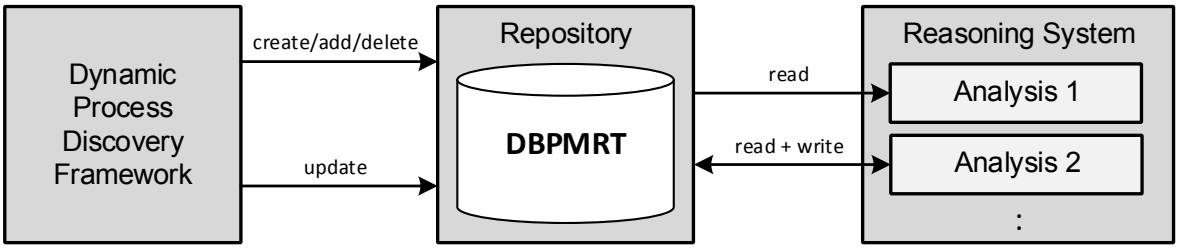


Fig. 6.2 Repository Architecture for the General Reasoning Framework

6.1.2 Event to Performance Processing (External Component)

For clarification it has to be stated at this point that the event to performance processing component discussed in this (sub-)section is not an analysis component based on a DBPMRT (as previously discussed). Instead it is based on processing BP events and is only used as a sub-component of the performance prediction analysis component (see next section). Essentially, the event to performance processing component is an individual Business Activity Monitoring (BAM) solution (see Section 2.2.3) that is originally used independent of the DPDF. As it represents an external tool and offers no contribution in the context of this thesis only its essential features are briefly discussed.

The event to performance processing component takes single events and aggregates them to selected PPIs in order to monitor their development. Objective-wise this is similar to what the agents for the DBPMRT performance perspective are calculating. The difference, however, is that the DBPMRT is only focussed on *external* PPIs such as *Path Probabilties* and *Process Instance Occurrence* whereas the performance processing component is focused on mainly *behavioural* PPIs such as *End-to-End Processing Time*, *Role Queue Length Resource Utilisation*[1]. Additionally, in the performance processing component the PPI calculation is not done in a dynamic fashion where older events have less influence on the current value (i.e. no ageing) but instead by using the more conventional method of sliding windows. Figure 6.3 shows how the performance processing component conceptionally operates for the example PPIs *Throughput*, *Processing Time*, and *Queue Length*:

- *Throughput* (top of the figure): The throughput is essentially counting the occurrences of a certain event for a given sampling rate and sampling interval. The first determining the rate at which a throughput value is recorded and the latter the time window for which residing occurrences are contributing to the throughput value. With the sliding window approach it is possible to have occurrences contributing to throughput values associated with different times, e.g. the orange occurrences are part of the yellow and red throughput values. Note, that the PPI *process instance occurrence* is a specialisation of the throughput PPI counting the start event occurrences of a BP.

[1] While *external PPIs* describe environment-dependent outside influences, *behavioural PPIs* describe the performance directly influenced by the characteristics of the BP.

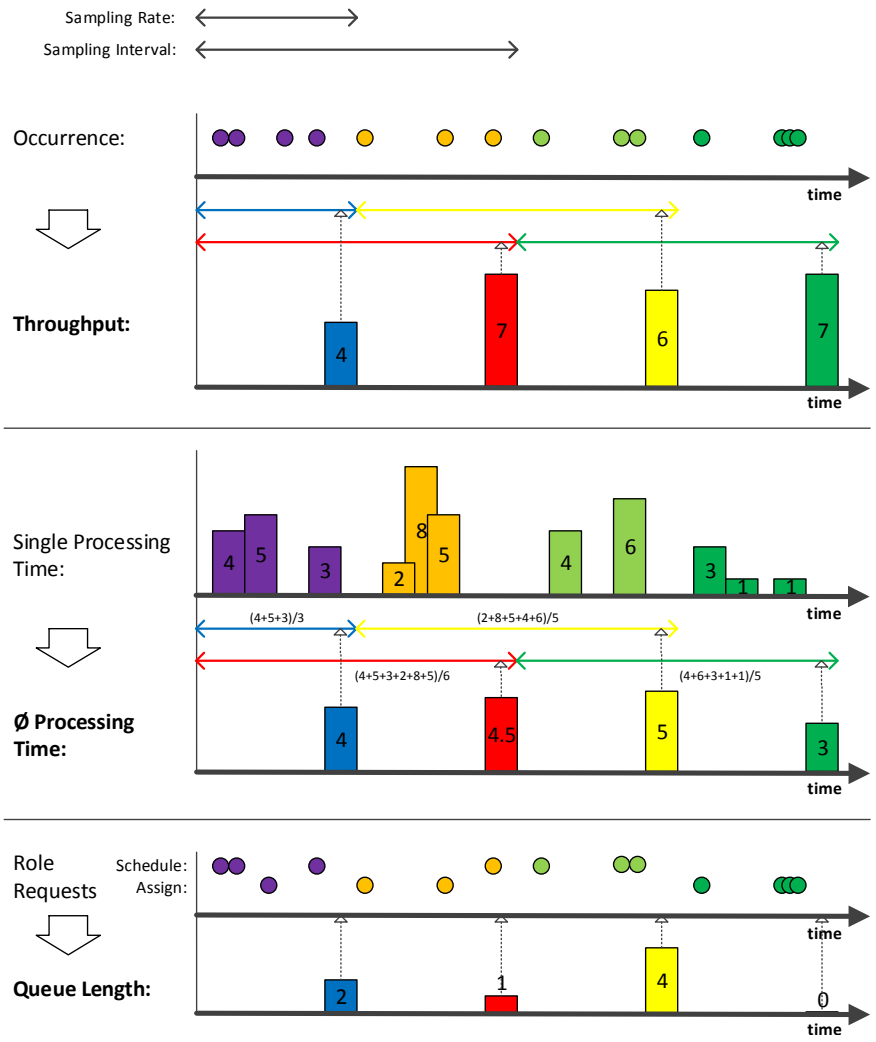


Fig. 6.3 Exemplary Calculation of PPIs Throughput, Processing Time, and Queue Length

- *Processing Time* (middle of the figure): The processing time is a measure that describes how much time has passed from a initial event to the target event, e.g. from start to end of a BP: *End-to-End Processing Time*. It is similarly computed like the throughput PPI with a specified sampling rate and interval. However, instead of counting occurrences, the average of the occurring single processing times is calculated and recorded, e.g. the yellow value is the average of the orange and light green recorded single processing times: $\frac{2+8+5+4+6}{5} = 5$.

- *Queue Length* (bottom of the figure): The queue length (of roles or activities) is a discrete state measure (unlike the former two) that is only defined by a sampling rate but not sampling interval. It essentially determines how many tasks are waiting to be processed by an available resource. This is determined by offsetting the number of occurrences of "schedule" events (a resource is requested) with that of "assign" events (specific resource has been assigned), e.g. the red value is: 2 (former queue length) + 1 (the one orange schedule) − 2 (the two orange assigns) = 1.

Other PPIs, such as *Resource* or *Role Utilisation* (see Section 2.5.1), are calculated similar to the described PPIs.

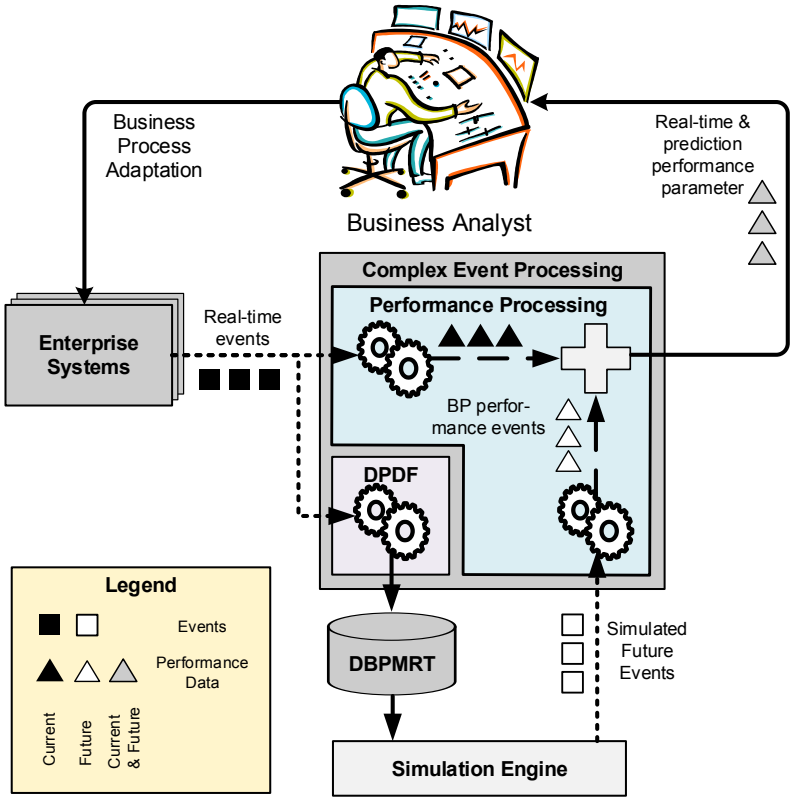


Fig. 6.4 Concept of Using Simulation for Performance Prediction

6.1.3 Performance Prediction via Simulation

In this section an analysis component (only read access) for the overall reasoning framework introduced in Section 6.1.1 is further specified for proof-of-concept purposes: the performance prediction via simulation. The potential benefits of simulation over statistical methods is discussed in Section 2.5.2, e.g. as opposed to standard statistical methods such as trend analysis a simulation tries to emulate the complex and interdependent structures of a system and not regard a performance measure in isolation (see Figure 2.22 on page 59). However, due to the fact that a simulation often only is a simplification of the real system, predictions of a further away future might turn out to be far from what is actually going to happen. Hence, the proposed performance prediction is a transient short-term prediction, i.e. predicting the immediate future development of PPIs taking the current reflective system state into account.

The methodology proposed makes use of another advantage of simulations: the ability to produce future events, which are of the same type as the live events generated by the actually executed business process. For this reason simulated future events can be processed by existing event stream analysis tools in the same way as the live events. The previously introduced performance processing component is such a tool and can in this way calculate the historic, current, and future values of a PPI. Figure 6.4 depicts the concept along with a schematic data flow and involved components[2] that provides performance

[2] Flows of raw data events like live and future events are depicted as fine grained arrows; processed performance data as coarse grained arrow; common data exchange via services as uninterrupted arrows

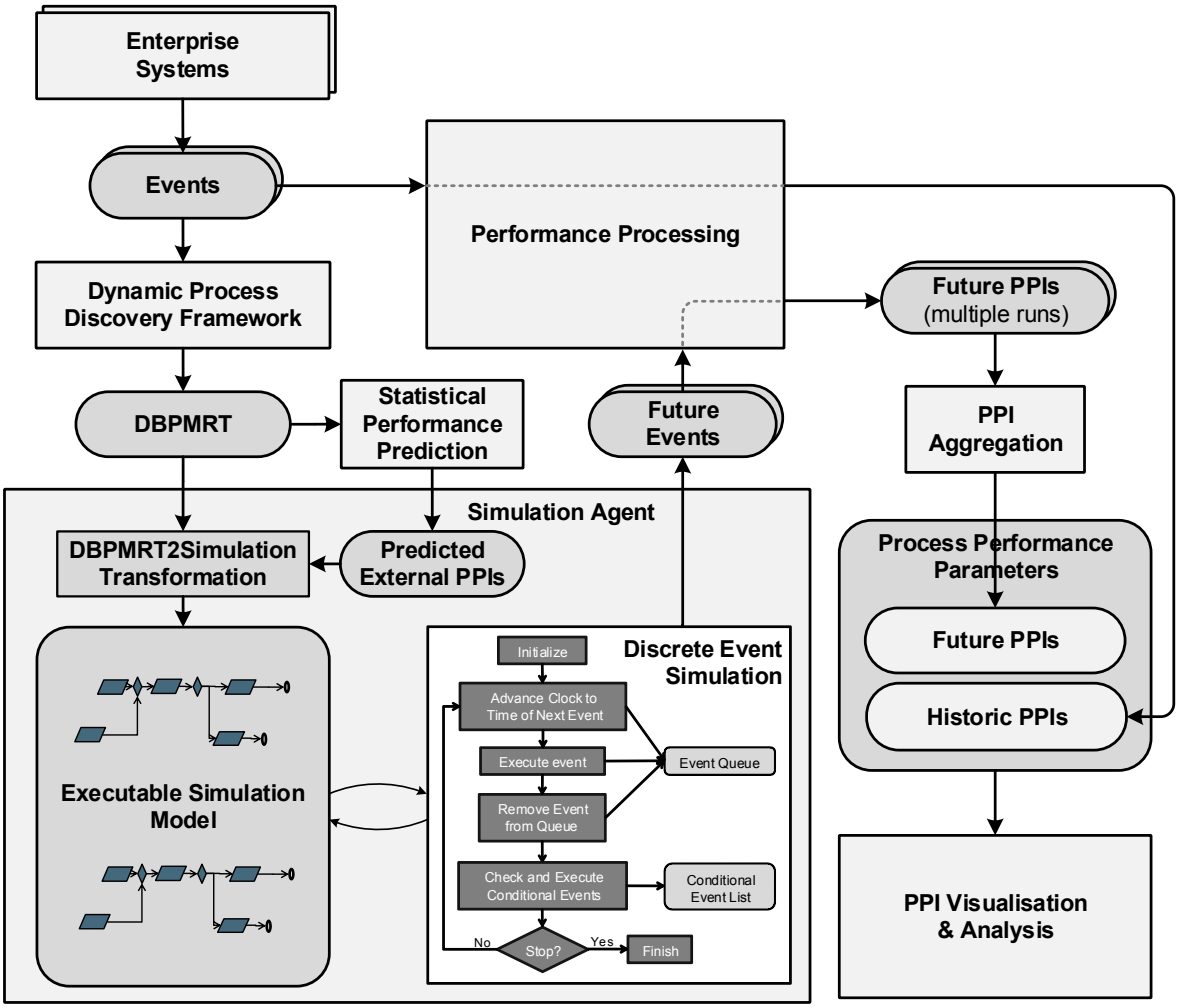


Fig. 6.5 Complete Information Flow for the PPI Prediction using BP Simulation

predictions via simulation with minimal adaptation effort on the part of the performance processing component. The enterprise systems continuously produce live events (black squares) which are processed and aggregated to real-time performance values (black triangles). Additionally, the live events are processed by the DPDF to the DBPMRT, reflecting the holistic state of the system. With this information short-term simulations are carried out producing future events (white squares). These future events are fed back into the performance processing component where, just like for the live events, they are aggregated to performance values (white triangles). The available historic, current, and future performance data is then merged (grey triangles) and further processed, e.g. by presenting the results in a dashboard, sending warning notifications, or triggering more complex events leading to system adaptations. Following this concept other event processing analysis techniques than performance aggregation can be integrated to not just support the analysis of the current (and historic) but also of the future states.

With regards to the use case of performance prediction via simulation this concept is further refined. The proposed information flow is shown in Figure 6.5 and features the following new models and agents:

- *Statistical Performance Prediction*: This component predicts the future development of *external* PPIs stored in the performance perspective of the DBPMRT. It uses regression algorithms to extrapolate future values from a set of historical data points.

- *Simulation Agent*: This component is in charge for orchestrating the transformation of a DBPMRT and the predicted external PPIs into simulated future events and consists of the following sub-elements: (1) A transformation from DBPMRT to an executable simulation model into which also the predicted external PPIs (necessary for a simulation) are incorporated; (2) The executable simulation model is a semantic enhanced copy of the DBPMRT that is interacting with the discrete event simulation; (3) The discrete event simulation is the component executing the simulation model via timed and conditional events (see Section 2.5.2). The simulation produces future events and runs until a specified date in the future is reached. Since the BP simulation is non-deterministic method it is executed a fixed number of times creating multiple batches of future events.

- *Future Events and PPIs*: The multiple batches of future events are respectively aggregated to future PPIs with the help of the performance processing component. The result is a set of PPI results projecting multiple possible futures.

- *PPI Aggregation*: The set of future PPIs are then aggregated to a single future projection representing the most likely/average outcome of the PPIs produced by the different simulation runs. The result of this aggregation is merged with the current and historic PPIs.

- *Process Performance Parameters*: It is the container element for historic (including current) PPIs calculated by the performance processing component and the future PPIs predicted by DBPMRT reasoning methodology explained earlier. This can be further processed by higher-level analyses (e.g. optimisations) or visualised (see Appendix C).

6.2 Case Study Scenario: Akron Heating Retailer

Akron Heating[3] is a retailer company with 28 active employees. It sells a wide range of products via telephone or online, i.e. it has no local shop. Dependent on the season and weekday usually a range of 100-300 orders per day are recorded. The management of *Akron Heating* wants to be able to react proactively to unforeseen peaks or gradual changes in demand and decides to tightly monitor and predict key performance indicators of all internal IT-supported BPs.

[3] Akron Heating is an artificial company emulating a real-life retailer scenario. It is used in SAP-internal projects to showcase products and prototypes.

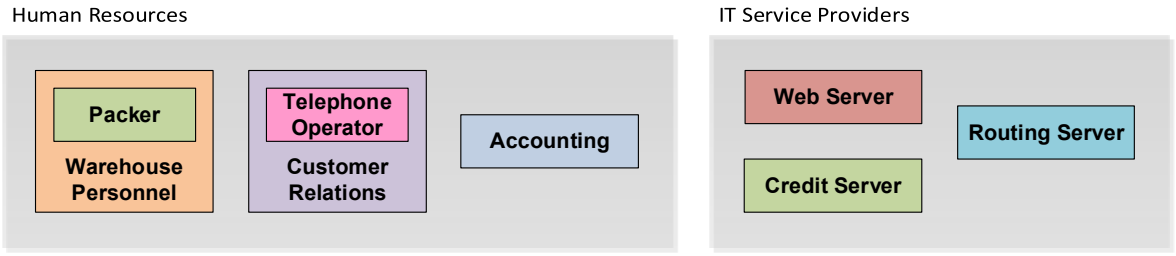


Fig. 6.6 Actively Participating Roles in the Akron Heating Company

6.2.1 Organisational Structure

The documented organisational structure (from BP point of view) of Akron Heating is shown in Figure 6.6: Employees (not management) have at least one of the five roles in the "Human Resources" box. Commonly, *Packers* are also used for tasks as *Warehouse Personnel* and *Telephone Operators* work also for *Customer Relations*. Employees with the role *Accounting* have no other relevant role in this scenario. Furthermore, the three "IT Service Providers" *Web Server*, *Credit Server*, and *Routing Server* are in charge for executing specific automated task.

6.2.2 IT-supported Business Processes

Akron Heating features six IT-supported BPs which are partly interconnected (see Figure 6.7): the four customer-related BPs *Create Telephone Order, Create Online Order, Process Order*, and *Return Item*, and two organisation-internal processes *Refill Stock* and *Expense Payment*. Some of the processes are used by others, i.e. they are sub-processes. For instance, in *Create Telephone Order* an operator is initiating an *Online Order Process*. However, that does not mean, they cannot be initiated from somewhere else than a parent process. For instance, *Create Online Order* and *Expense Payment* are both, sub-processes of another BP and self-initiatable. In fact, *Process Order* is the only BP that is only used as sub-process (by *Create Online Order* and itself). *Create Telephone Order*, *Return Item*, and *Refill Stock* are the only processes not used as sub-processes.

The BPs associated with processing an order are *Create Telephone Order* (see Figure 6.8), the *Create Online Order* (see Figure 2.2 on page 14), and *Process Order* (see Figure 6.9).

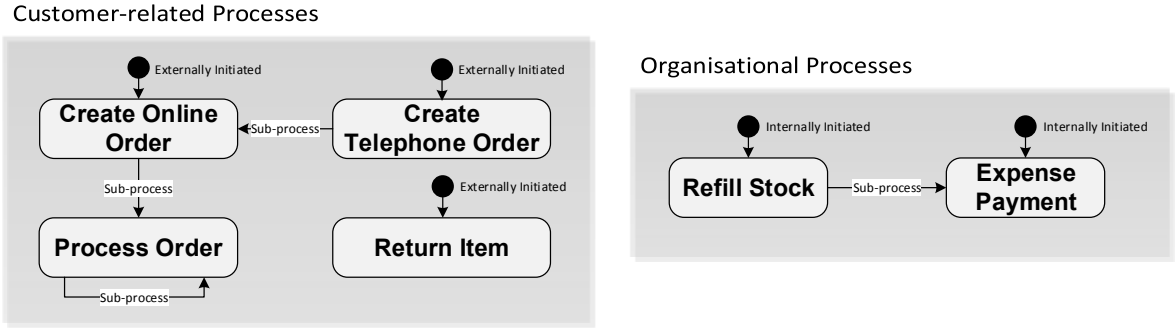


Fig. 6.7 IT-supported BPs in the Akron Heating Company

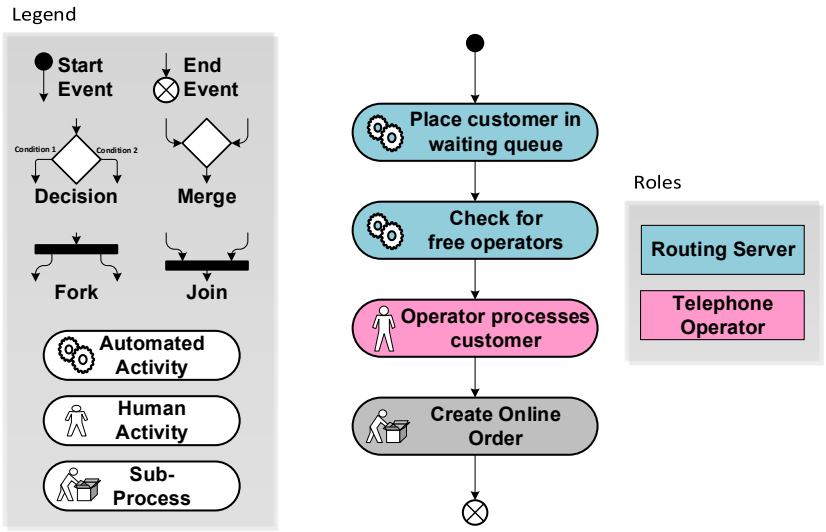


Fig. 6.8 The Planned "Create Telephone Order" BP in the Akron Heating Company

The remaining three BPs not directly associated with the ordering process are shown in Appendix D. As Figures 6.8 and 6.9 show, the respective BPs can be diverse with regards to their complexity: While the *Create Telephone Order* is a simple sequence of four activities (one of them the *Create Online Order* sub-process), *Process Order* is relatively complex featuring itself as a recursive sub-process. The arguably most complex BP in the Akron Heating scenario is *Return Item* shown in Figure D.3 on page 237.

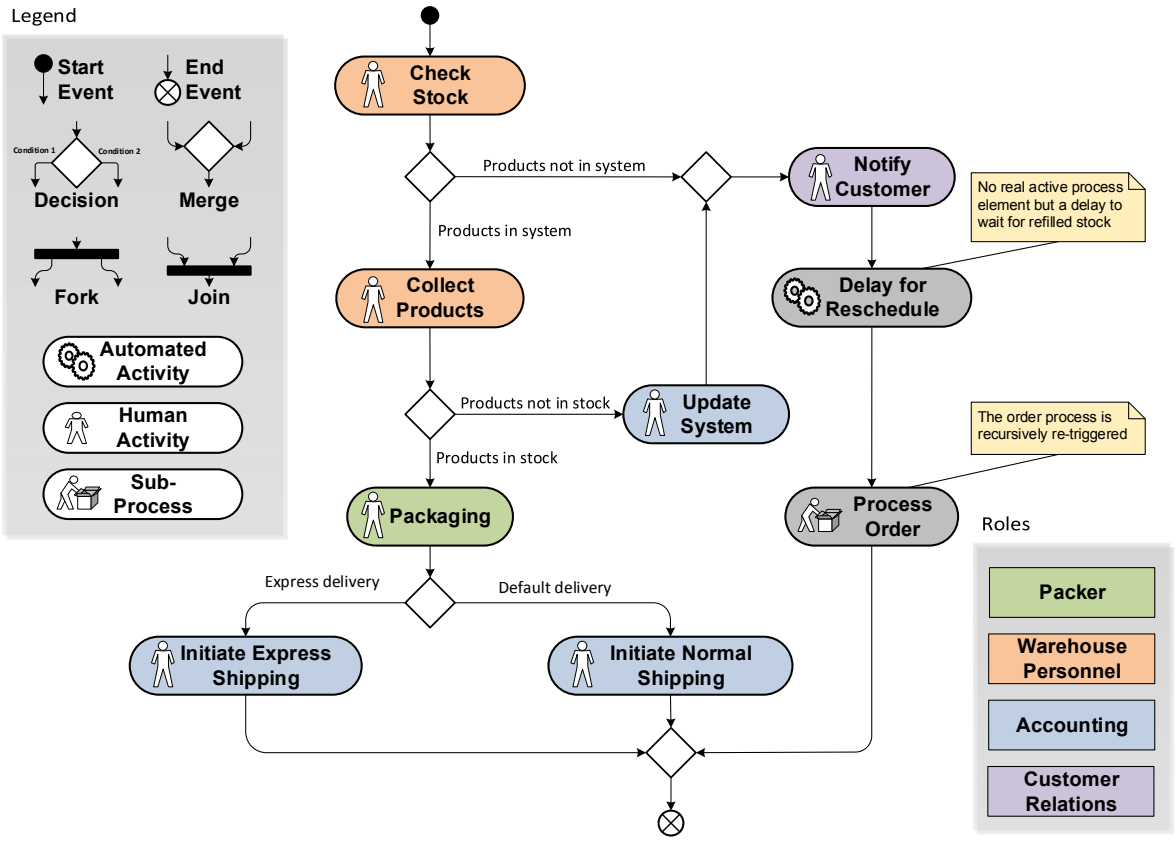


Fig. 6.9 The Planned "Process Order" BP in the Akron Heating Company

6.3 Analysis Results

In this section it is investigated if run-time reasoning can be improved by utilising DBPMRTs (see fourth research question in Section 1.4). This will be done for the reasoning use case of performance prediction and evaluated in the context of the Akron Heating Retailer case study. Since all proposed contributions are involved in the reasoning methodology this represents not only an evaluation of the proposed predictive reasoning framework but also for the DBPMRT reference model as well as methodologies and algorithms employed in the DPDF. The experiment is set up as follows:

- DPDF configuration: The core CCM was configured with its default parameters (see Section 5.8.1) and the ageing method of *discrete ageing* with *trace influence factor* $t_{if} = 0.002$ and *interpretation frequency* $m = 10$ was used.
- Approximately eight weeks of event data produced by the Akron Heating were recorded (379200 events; ≈ 6771 events per day) and later replayed as event stream (in higher speed). One month of that time (216088 events) was used to "warm-up" the footprints without any interpretation or simulation. The remaining 163112 were then analysed with the full DPDF and performance prediction framework.
- The statistical method used for predicting the future of the external PPIs in the DBPMRT (see Statistical Performance Prediction in Figure 6.5) is the *periodic trend* extrapolation which is later explained in Section 6.3.3. Other prediction-related experiment configurations are detailed in that section, too.

The following two Sections 6.3.1 and 6.3.2 discuss the (last state of the) discovered DPBMRT with regards to the control-flow and resource perspective. The shown models are screen shots of the DBPMRT visualisation tool (see Appendix Section C.1). The performance prediction results and a comparison to other statistical prediction methods is then presented in the final Section 6.3.3.

6.3.1 Roles and Resources

During the experiment three changes in the resource perspective were noticed which were, however, of a minor nature, e.g. one resource was given an additional role. Figure 6.10 shows the resource perspective of the DBPMRT at the end of the experiment. Annotated in red are the official names of the respective roles (smaller rounded rectangles). As established in Section 5.5.2 those names are usually not transmitted by the events which is why the roles have to be discovered. Their representative names are the set of activities they are representing. Some of the roles are discovered as planned and presented in the scenario description, e.g. *Telephone Operator* (5 employees) and *Customer Relations* (3), including the fact that all of the former have at the same time the latter role. Also, all IT service providers and other non-resource activities are represented with roles

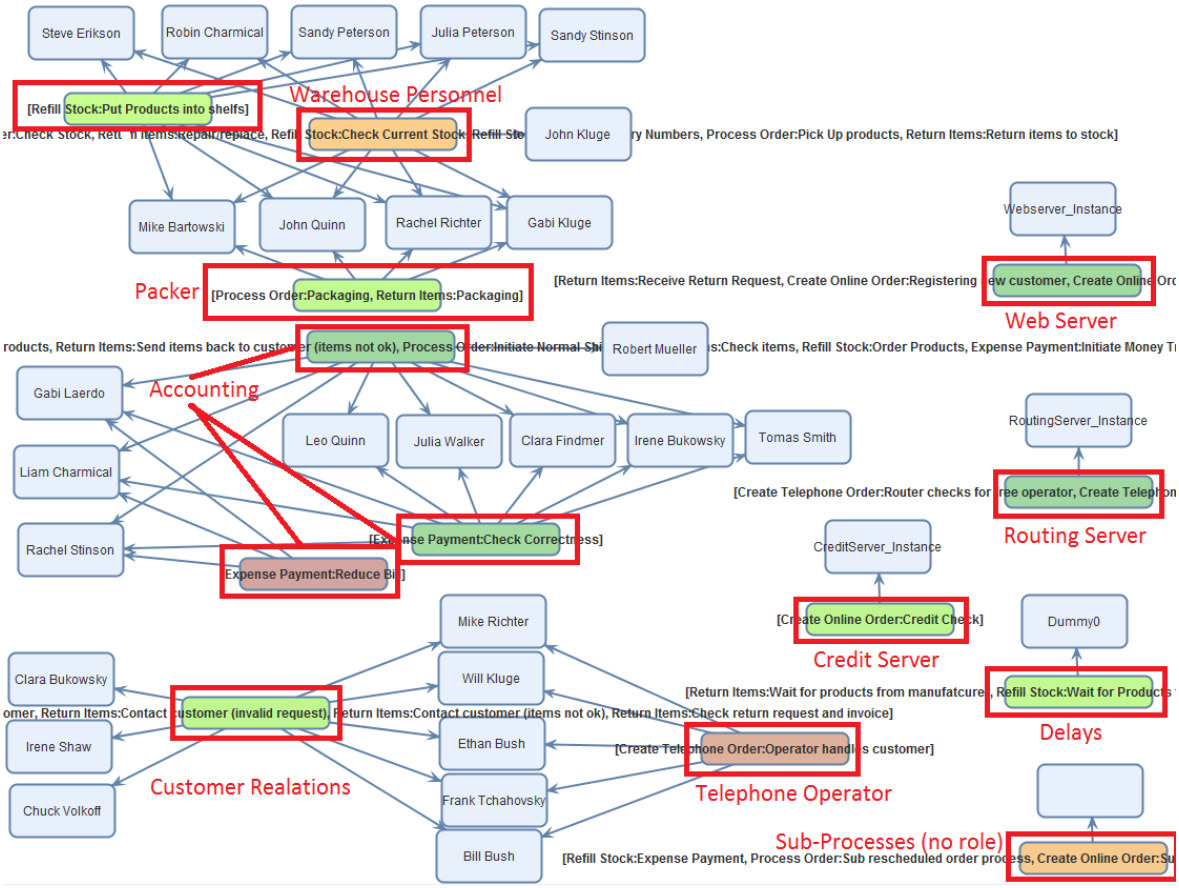


Fig. 6.10 Discovered Resource Perspective

in the picture. The planned "Accounting" role consists of three different representatives in the DBPMRT, each representing a different set of activities. All 9 accountants have at least one association with one of the three representatives and no other role of a different kind. However, not all accountants work on the same activities. This can be due to a specialisation of employees since the role of accountant is originally very diverse and involves (too) many different activities. A similar effect is observable with *Warehouse Personnel* where "John Kluge" is not in charge for activity "Put Products back into shelfs" which is why two separate roles exist in the model. The dependency that all *Packers* have additionally the role of *Warehouse Personnel* is reflected in the model and in the system.

6.3.2 Business Processes

Figure 6.11 shows the control-flows of the three processes involved with the ordering in the DBPMRT at the end of the experiment. The control-flow visualisation also shows the performance values (i.e. (*external*) PPIs for simulation) annotated to the respective BP elements, e.g. "ANT" = *Activity Networking Time*, "PIO" = *Process Instance Occurrence*. Manually annotated by the author in black colour are the names of the three involved BPs and the boxes separating them from each other. Furthermore, light blue arrows represent associations between a parent process and its sub-process, e.g. between "Sub Order

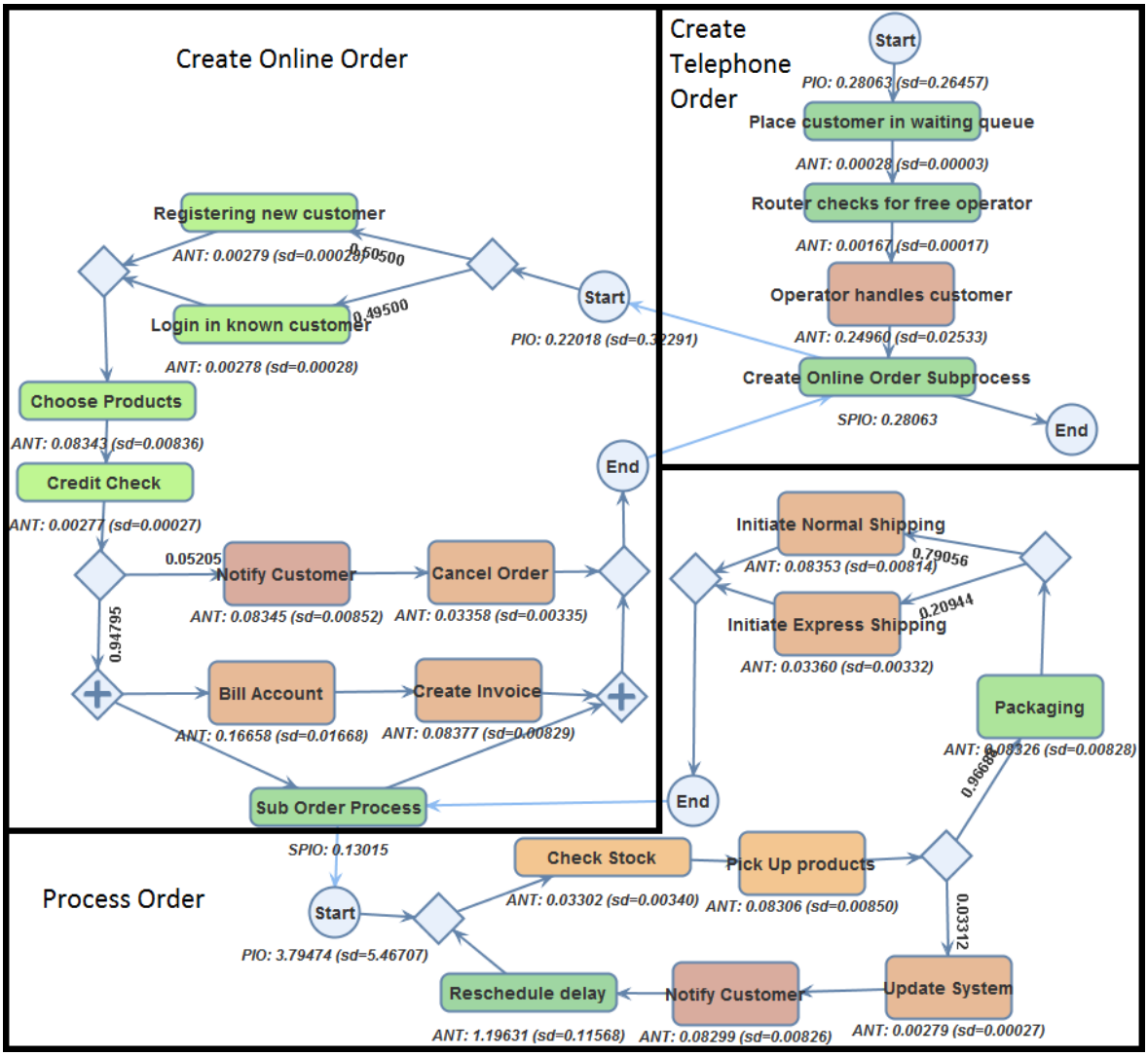


Fig. 6.11 Discovered Control-flows of *Create Telephone Order*, *Create Online Order*, and *Process Order* BPs

Process" and the BP start and BP end elements of the "Process Order" BP. While human activities are depicted as big rounded rectangles, sub-processes and automated activities are depicted as slim rounded rectangles. The colour of each activity is that of the associated role[4] (see Figure 6.10).

The discovered control-flows of the "Create Telephone Order" and "Create Online Order" BPs are identical to their respective planned models introduced with the case scenario (see Figures 6.8 and 2.2). This indicates that the system implementation of these BPs did not majorly deviate from the planned scenario as well as no concept shifts and only a few exceptions occurred during run-time. This is not the case for the "Process Order" BP: Here, the planned recursive structure (i.e. calling itself as sub-process) is dissolved into a loop (see bottom of figure). After investigating the events the implementation followed the planned recursive structure, the DPDF interpreted this behaviour into an imperative

[4]The colours were randomly chosen by the DBPMRT visualisation tool which has the unfortunate effect that some shades of green are difficult to distinguish.

loop construct. Furthermore, the optional path from "Check Stock" to "Notify Customer" does not seem to appear in the real-life log indicating either a deviation from the planned model or that the condition for that path never applies in the system.

The discovered control-flows of the other BPs that are not directly connected to the ordering process are shown in Appendix Section E.1 starting on page 238.

6.3.3 Performance Predictions

In this section the proposed performance prediction reasoning methodology is evaluated against a statistical prediction method of extrapolating from a regressed function. That means, given a set of historical data points, a function type is regressed (parametrised) so that the *Mean Squared Error* (MSE) between function data points f_i and actual data points r_i is minimal, i.e. minimal $MSE = \frac{1}{n}\sum_{i=1}^{n}(r_i - f_i)^2$. The resulting parametrised function is then used to predict data points for "future" values. Two function types are regarded for this method:

- *Trend*: This is the equivalent of a trend analysis, i.e. applying a simple linear regression[5] to parametrise a_0 and a_1 for the linear function

$$f(x) = a_1 * x + a_0$$

- *Periodic Trend:* This function type additionally adds a sinus element to account for periodic developments. The Levenberg-Marquardt algorithm[6] was applied to parametrise a_0, a_1, a_2, and a_3 for the function

$$f(x) = a_3 * sin(a_2 * x) + a_1 * x + a_0$$

Figure 6.12 shows the real value data points (black solid line) and the two functions which were regressed from the real values: Trend (light blue solid line) and Periodic Trend (dark blue solid line). The dashed lines of the same colour represent the extrapolated data points of the respective functions and thus the predicted future development.

For the case study experiment the external performance processing component (see Section 6.1.2) is configured with a sampling rate of 1 hour and a sampling interval of 10 hours. Furthermore, the predictions were carried out roughly every 24 hours, always predicting the next 24 hours. The two statistical prediction methods take the last 30 hours of data points for regressing the respective functions. The simulation-based method proposed in this chapter takes for each prediction interval, the current state of the system, i.e. the DBPMRT. Since it is a simulation- and probability-based method it is non-deterministic

[5] for this `org.apache.commons.math3.stat.regression.SimpleRegression` was used

[6] for this `org.apache.commons.math3.optimization.general.LevenbergMarquardtOptimizer` was used

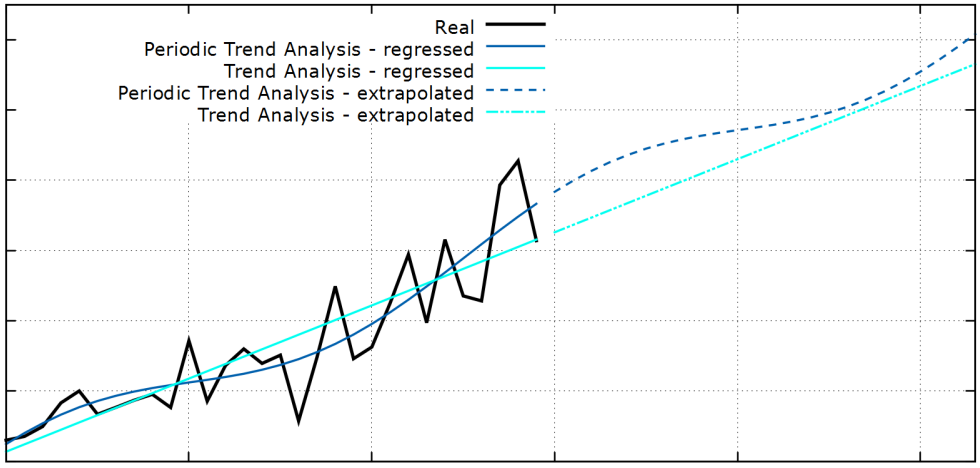


Fig. 6.12 Example Extrapolation on Regressed Functions: *Trend* and *Periodic Trend*

and thus will be executed 20 runs for each prediction and which are aggregated to an average value representing the predicted PPI. In particular, the predictive quality of the simulation methodology and the statistical methods is evaluated for the PPIs *Queue Lengths* of the roles "Telephone Operator" and "Accounting" as well as the *End-to-End Processing Time* for the "Create Online Order" BP in the following. The results for some additional (but not further discussed) PPIs are presented in the Appendix Section E.2. The discussed and additional PPIs are further analysed in a summary section where an overall accuracy is determined from these results for the individual prediction methods.

Queue Length: Telephone Operator

In Figure 6.13 an excerpt of the predictions of each of the methods in relation to the really occurring queue length values is shown. 6 prediction episodes are shown in the picture

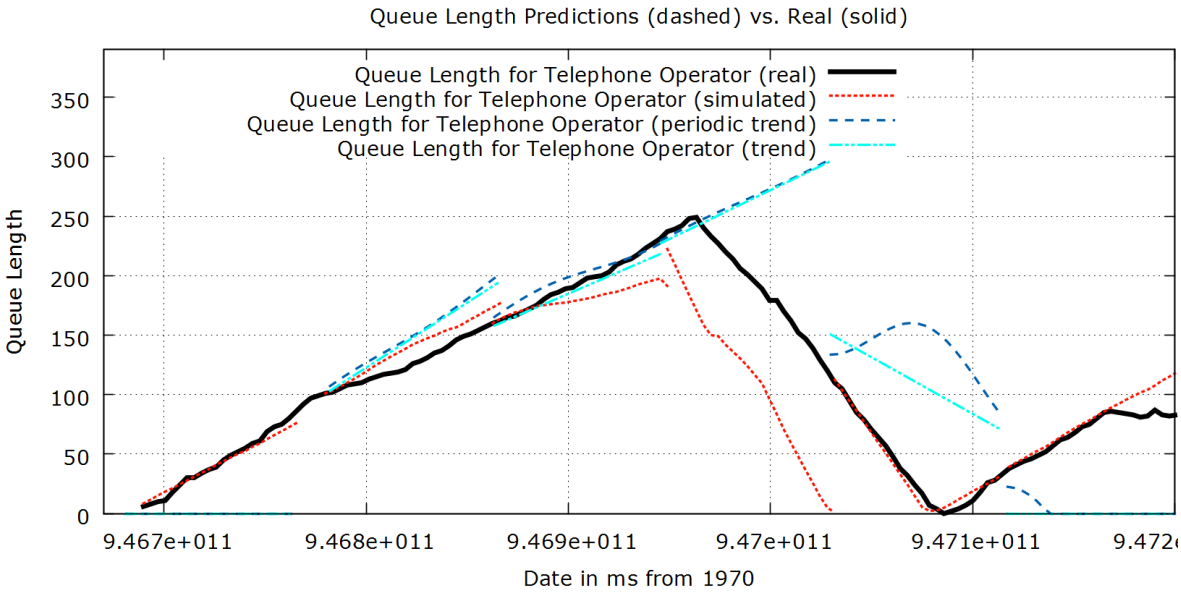


Fig. 6.13 Six Episodes of Prediction Results vs. Real Development for *Queue Length* of "Telephone Operator" Role

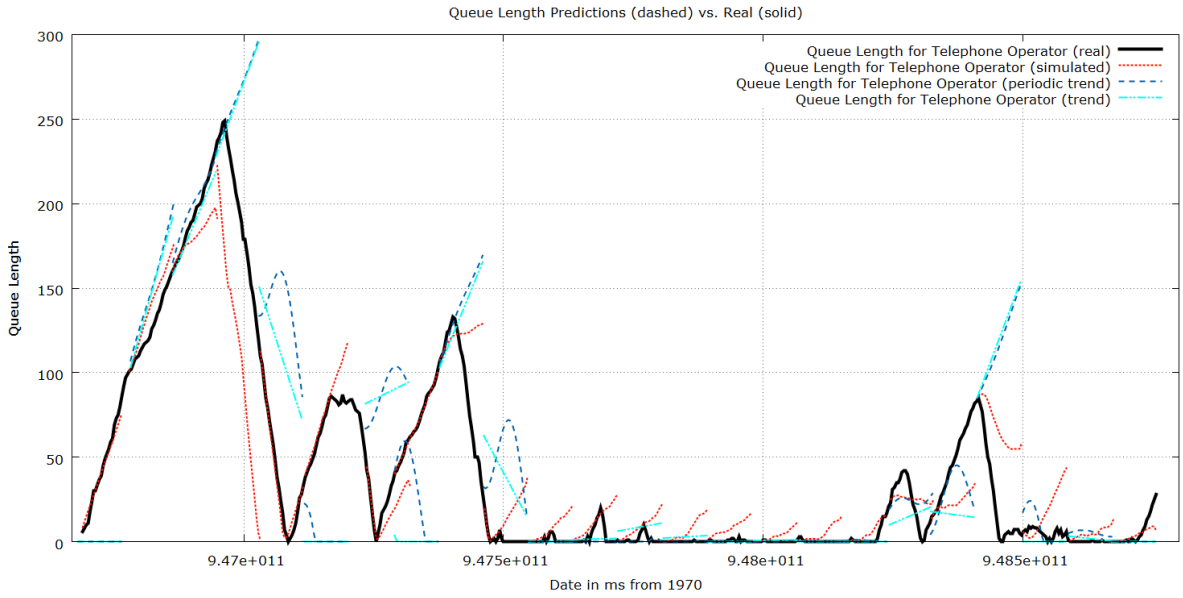


Fig. 6.14 Prediction Results vs. Real Development for *Queue Length* of "Telephone Operator" Role

(highlighted by breaks between the predicted lines). Whenever a trend continues all prediction methods are predicting this to continue, especially in episodes 1, 2, and 3. In contrast, the individual predictions for episodes 4, 5, and 6 diverge significantly: While the simulation does predict the general direction, sometimes even the correct inflection point (episode 5), the statistical methods are generally further off. A complete overview of all prediction results of the different methods for the entire 25 days of the scenario is shown in Figure 6.14. Here it can be seen, that the simulation predicts the inflection points very well but performs less well when the development is flat (middle of the picture).

If the respective predictions are compared to the real value an error can be computed. If this is done for all prediction episodes the MSE can be computed depending on how far ahead is predicted, e.g. one hour ahead the prediction error of method x was better than the prediction error of method y. Figure 6.15 shows the MSE of each of the prediction methods: The values become less reliable, i.e. the error increases, the further ahead it is predicted. The figure shows quantitatively that the simulation method is significantly better suited to predict the queue length of the "Telephone Operator" role than the statistical methods.

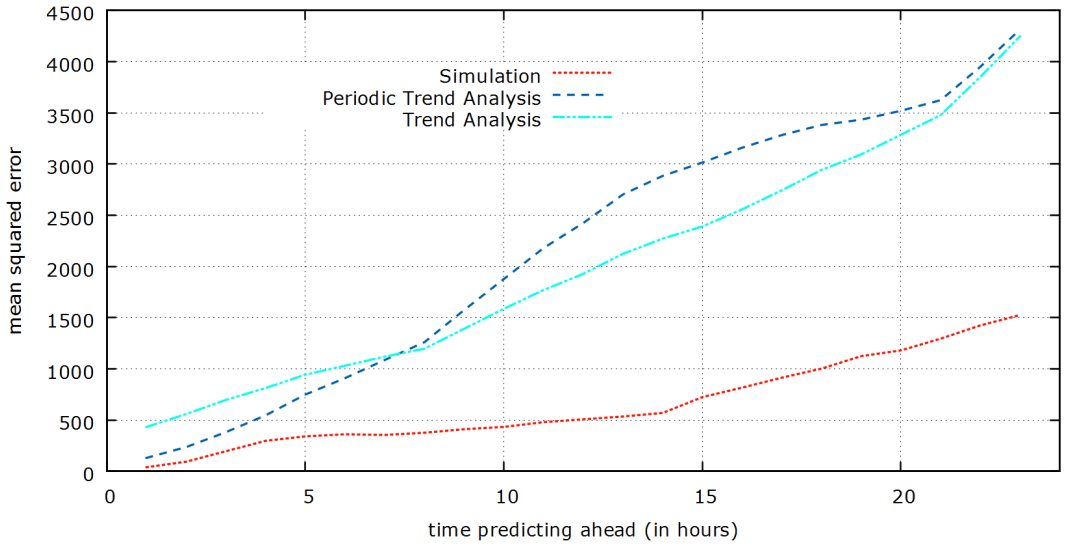


Fig. 6.15 MSE of the Prediction Methods for *Queue Length* of "Telephone Operator" Role

Queue Length: Accounting

Figure 6.16 shows the real *Queue Length* values of "Accounting" as well as the prediction results of the different runs/episodes. The break in the black line is due to a change in the resource perspective which was related to "Accounting" role. The development of this role queue length is a lot more diverse since it covers many activities and with strong interdependencies to other developments in the BPs. Generally, it can be observed that, similarly like for the "Telephone Operator" the simulation prediction tends to be better but is sometimes also not very close. The overall performance of the different methods with regards to the MSE is shown in Figure 6.17: While still predicting "twice" as good as the other both methods the simulation prediction does not perform significantly better than the others for the "Accounting" queue length.

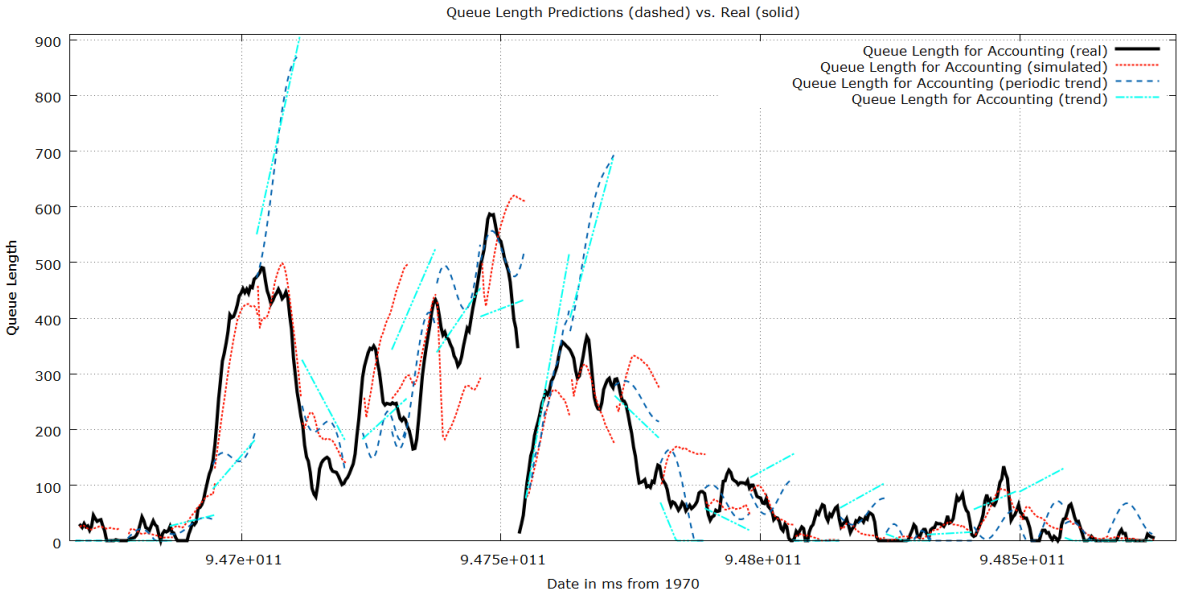


Fig. 6.16 Prediction Results vs. Real Development for *Queue Length* of "Accounting" Role

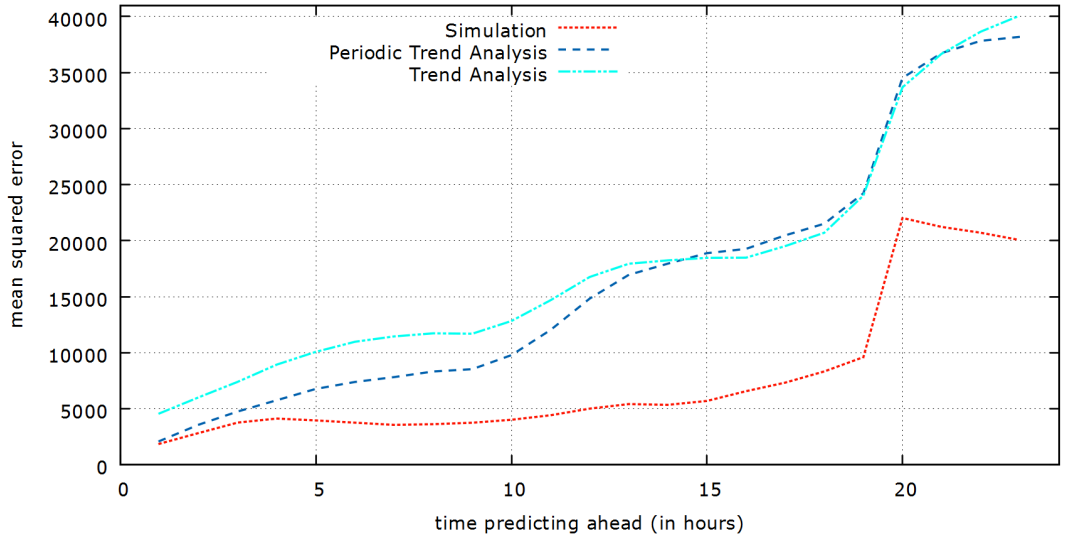


Fig. 6.17 MSE of the Prediction Methods for *Queue Length* of "Accounting" Role

End-to-End Processing Time: Create Online Order

Figure 6.18 shows the real *End-To-End Processing Time* values of "Create Online Order" as well as the prediction results of the different runs. Despite being influenced by many internal system developments this PPI is less fluctuating than the "Accountant" queue length. On average the simulation seems to predict roughly the correct development in a bit more than half the episodes while this is less so the case for the statistical methods. This subjective impression is confirmed in Figure 6.19 which shows that the MSE of the simulation prediction is significantly better than that of the other two. It even stays relatively flat for longer term predictions which indicates that a lot of long term effects are influencing this PPI.

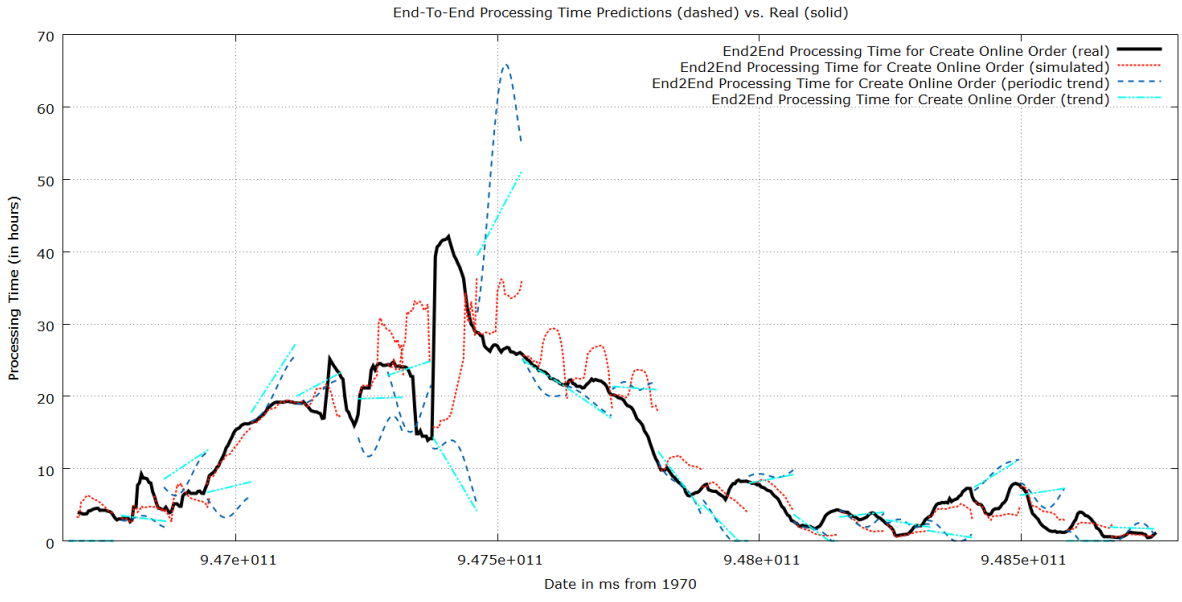


Fig. 6.18 Prediction Results vs. Real Development for *End-to-End Processing Time* of "Create Online Order" BP

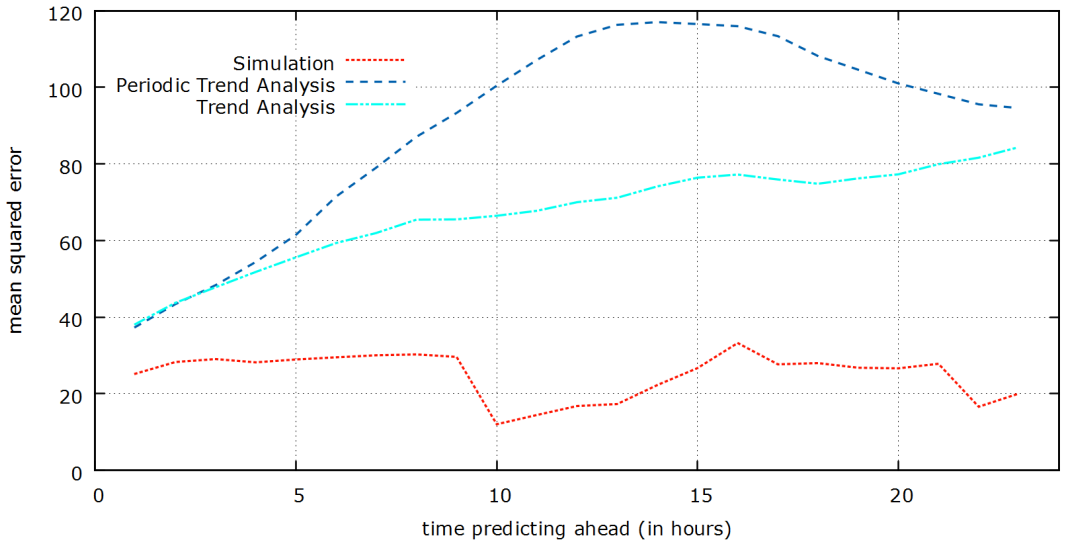


Fig. 6.19 MSE of the Prediction Methods for *End-to-End Processing Time* of "Create Online Order" BP

Discussion of PPI Prediction Results

The case study evaluation results show that the simulation-based prediction based on the DBPMRT improves the prediction results for the regarded PPIs when compared to common statistical prediction methods, i.e. *trend* or *periodic trend* analysis. In many cases the results were significantly better, in particular for the predictions of the "Telephone Operator" queue length (see Figure 6.15), the "Create Online Order" BP end-to-end processing time (see Figure 6.19), the "Return Items" BP end-to-end processing time (see Figure E.5), and the "Packer" utilisation (see Figure E.7). For other predictions, i.e. the "Accounting" queue length (see Figure 6.17) and the "Initiate Express Shipping" activity throughput (see Figure E.9), the predictions were moderately or even only slightly better than the statistical methods. Overall, the predictions of the simulation method are more often close to reality than deviating from it.

Nevertheless, the results show that large discrepancies between the simulation predicted values and the really measured values exist for some cases, e.g. for the "Accounting" queue length in the centre of the time series in Figure 6.16 two or more prediction runs even show the wrong trend (the predictions go upwards but the reality goes downwards). These discrepancies can have a multitude of reasons of which three are listed in the following:

- The case scenario can be very volatile with many peaks and lows in demand or resource availability. For such a scenario any prediction will be difficult since too many hidden factors are involved.
- Structural dependencies also play an important role regarding the predictability: If, for instance, an element at the beginning of the process is monitored, then mostly only external (e.g. business process instantiation) but not BP-dependent influences

(e.g. resource availability) drive the performance parameter. For the prediction of these PPIs BP simulation might not be of much benefit.

- The DBPMRT representation may be too much of a simplification/generalisation of the real system. For instance, the "Initiate Express Shipping" activity throughput is directly behind a decision which is abstractly captured by probability values in the DBPMRT. In contrast, the decision in the real system is based on instance-specific data. This simplification can be the reason for the observed discrepancy between predicted and real values for this PPI. The "Accounting" queue length could also be explained by a simplification of the DBPMRT: Since the "Accounting" role is involved in many different activities, the reason for the deviation could be the simplified representation of the resource perspective, e.g. all resources have the same efficiency and fulfil a role to 100%.

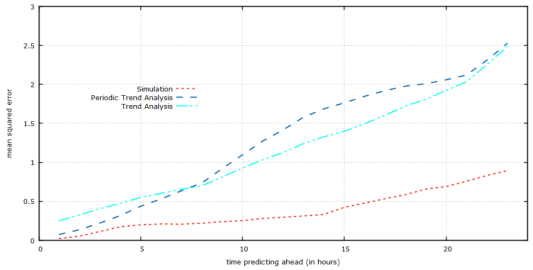
While the first two possible reasons are independent of the proposed solution, the latter can be mitigated and is the subject of future work (see Section 7.2).

Overall Accuracy

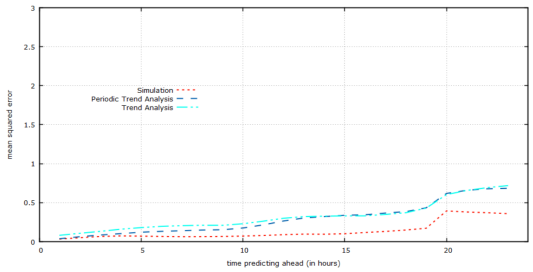
To give an overall assessment of the accuracy of the respective prediction methods, the error values (mean squared errors) from the three specifically discussed PPIs (see Figures 6.15, 6.17, and 6.19) as well as from the three additional PPIs in the appendix (see Figures E.5, E.7, and E.9) are normalised with the square of the PPIs' mean value so that they are in a similar range, i.e. normalised mean squared error $|MSE_{PPI}|$ is calculated by $|MSE_{PPI}| = \frac{MSE_{PPI}}{\phi^2_{PPI}}$, with PPI being one of the six PPIs: $PPI \in \{QLTelephoneOperator, QLAccounting, E2EOnlineOrder, E2EReturnItem, UtilPacker, TPExpressShipping\}$. The average values are:

$\phi_{QLTelephoneOperator} = 41.3461$, $\phi_{QLAccounting} = 236.2043$, $\phi_{E2EOnlineOrder} = 11.7104$,
$\phi_{E2EReturnItem} = 32.5625$, $\phi_{UtilPacker} = 0.2537$, $\phi_{TPExpressShipping} = 1.4779$

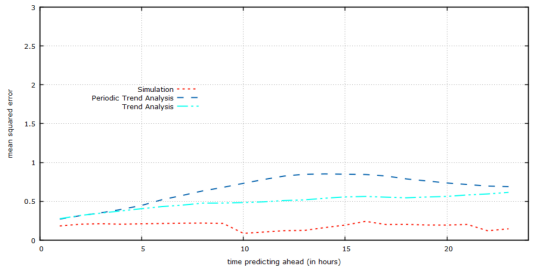
The normalised mean squared errors of each PPI and prediction method can be seen in Figure 6.20: The range goes from 0 to 3 in all sub-figures. One can see that for some calculations like the *End-to-End Processing Time* or "Return Items" BP the error is relatively small since the PPI does not fluctuate a lot and the predictions are quite close to reality (see Figure E.4). On the other hand, the prediction error is relatively for the PPIs *Queue Length* of "Telephone Operator" Role, *Utilisation* of "Packer" Role, and *Throughput* of "Initiate Express Shipping" Activity. These normalised prediction errors are now averaged to evaluate how well the prediction methods perform overall (see Figure 6.21). One can see that the proposed prediction via simulation based on the current DBPMRT far outperforms the statistical predictions trend analysis and periodic trend analysis. Very good results are achieved for predicting the more immediate future: Up to 11 hours in advance the simulation records only small errors, which later doubles but still stays far below the prediction errors of the statistical methods. This provides evidence, that the reasoning on the DBPMRT can yield more accurate results, i.e. the third objective of this thesis



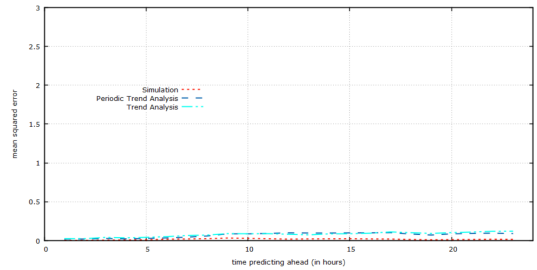
(a) *Queue Length* of "Telephone Operator" Role



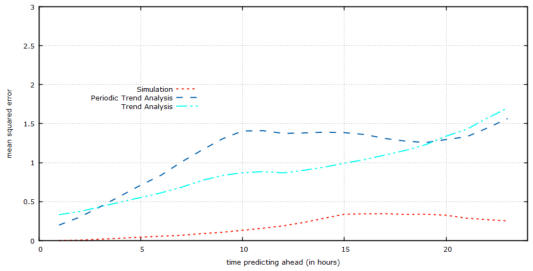
(b) *Queue Length* of "Accounting" Role



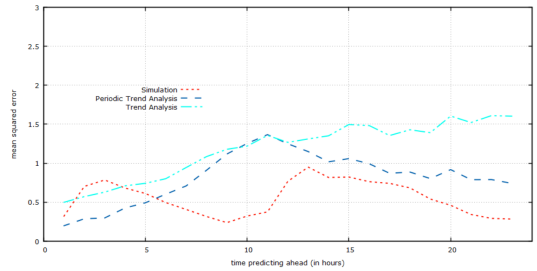
(c) *End-to-End Processing Time* of "Create Online Order" BP



(d) *End-to-End Processing Time* of "Return Items" BP (see Figure E.5)



(e) *Utilisation* of "Packer" Role (see Figure E.7)



(f) *Throughput* of "Initiate Express Shipping" Activity (see Figure E.9)

Fig. 6.20 Normalised Mean Square Errors of the Prediction Methods for the individual PPIs

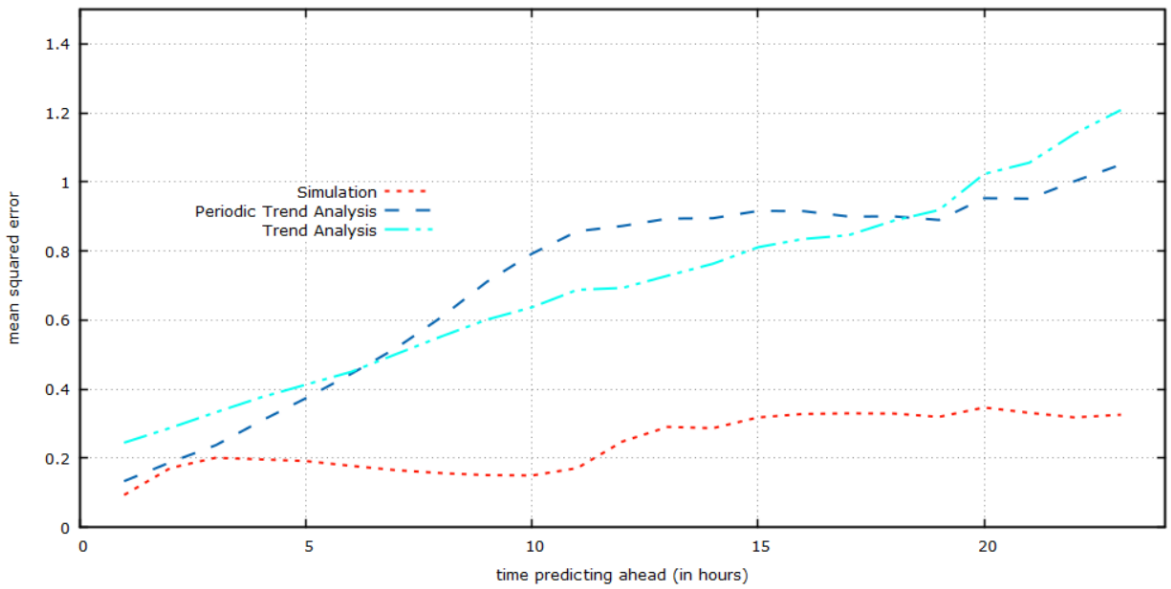


Fig. 6.21 Average Normalised Mean Square Error for the three different Prediction Methods: (1) Simulation, (2) Trend Analysis, and (3) Periodic Trend Analysis

has been successfully addressed. Furthermore, the DPDF addresses the gap identified in Section 3.4: It provides a solution to predict PPIs based on abstracted BP type level and instance level information from an event stream with noisy, incomplete, and changing behaviour.

6.4 Summary

This chapter addressed the third objective of reasoning on the abstracted run-time state of BPMSs, i.e. the DBPMRT. It presented a general reasoning framework (which can be regarded as MRT system by itself) that employs a repository architecture with the DBPMRT at the centre providing synchronised access for the update and analysis agents. Furthermore, a particular concept for a predictive reasoning was presented that uses BP simulations and integrates an external performance processing component to carry out performance predictions. The core principle of this reasoning methodology is based on the fact that simulation events and live events conform to the same format and can thus be similarly processed by an external event processing analysis, e.g. detection of future bottlenecks, what-if analysis, performance analysis. Following this approach these external event analysis tools enable *proactive* decision support on future states of the system.

In the remainder of the chapter a case study is carried out to evaluate the reasoning framework and by extension the DBPMRT and the DPDF: A case study scenario, the Akron Heating retailer company, was described in Section 6.2 which was then used to evaluate the overall concept. The analysis results with regards to the actually monitored control-flow and resource perspective as well as the prediction for selected PPIs were shown in Section 6.3. It shows that the simulation-based prediction based on the DBPMRT in most of the cases significantly and in some cases moderately or slightly improves the prediction results when compared to common statistical prediction methods.

Chapter 7

Conclusion and Future Research

This thesis has explored the potential of adopting and enhancing principles and mechanisms from the models@run.time domain to the business process domain for the purpose of run-time reasoning, i.e. it investigated the potential role of *Descriptive Business Process Models at Run-time* (DBPMRTs) in the business process management domain. This included (1) a gap analysis and extensive state-of-the art review of relevant work in the fields of Business Process Management and Model-Driven Engineering, as well as (2) identifying characteristics and proposing a reference specification for DBPMRTs, (3) providing an overall framework and associated algorithms to establish and maintain a causal link from an enterprise system to the DBPMRT in a dynamic run-time environment, and (4) proposing a general reasoning framework and a detailed concept for the use case of performance prediction. Overall, the thesis provided a novel approach that mitigates the lack of immediate causality between system and BP models, promotes a more dynamic handling of BPs, and enables reasoning on run-time data.

7.1 Results of the Thesis

7.1.1 Research Questions Revisited

How can the target system (BPMS) be effectively represented at run-time?

A BPMS can be effectively represented at run-time by a DBPMRT which are required to capture change on the levels of *dynamism*, *variability*, and *reflectivity*, as well as support the general *expressibility* of BP-domain models. Furthermore, two different types of DBPMRTs exist: a state and an evolution representation. The thesis provided reference meta-models which specify languages for these two types of DBPMRTs and fulfil the above requirements.

To what extent can a causal link from BPMS to BP model be established given that event data and BP model conform to different levels of abstraction?

Bridging the abstraction gap between BPMS and BP model to establish a causal link (from BPMS to BP model) is a multi-goal optimisation (fitness vs. precision vs. generalisation vs. simplicity) which is addressed by process discovery algorithms. This thesis proposed a novel algorithm to bridge the particularly challenging gap between BPMS emitted events and the control-flow of a BP: the *Constructs Competition Miner* (CCM). Additionally, algorithms to bridge the gap between events and other BP perspectives were proposed in the context of online process discovery, e.g. for resource, performance, instance state.

To what degree can a causal link from BPMS to BP model be maintained in a dynamic run-time environment?

The causal link in a dynamic run-time environment is difficult to maintain in real-time due to the big abstraction gap (algorithms have a high run-time cost), the dynamic nature of the BPs (frequent changes on the type level), and the very high frequency of events emitted by the enterprise system (potentially 1000s per second). To address these challenges this thesis proposes the Dynamic Process Discovery Framework (DPDF). The core features of the framework are the separation of the discovery methodology into two different lifecycles and the introduction of a *Dynamic Footprint* as an intermediate computer-oriented run-time model that abstracts event information: (1) The first lifecycle, synchronised with the event lifecycle, processes events to updates in the Dynamic Footprint in near-constant run-time. It uses the concept of *ageing* (discrete or time-based) allowing to "forget" old and "learn" new relations and thus abstractly recognise change on the type level; (2) the second lifecycle interprets the abstract Dynamic Footprint into the human-oriented run-time model, the DBPMRT. The interpretation is executed periodically or on demand and has less restrictions with regards to its computational run-time.

Can the quality of run-time reasoning be improved by utilising DBPMRTs?

To answer this question the thesis proposed a reasoning concept which is based on a DBPMRT and specialised for performance prediction via simulation. It furthermore introduces a case scenario in the context of which the reasoning framework and overall concept is evaluated. The case study demonstrated that the quality of run-time reasoning yielded more accurate results for the use case of short-term performance prediction. The more accurate prediction of near future developments promises better adaptive reasoning capabilities and shortens the BPM lifecycle.

How and to what extent can a target system emitting low-level events be causally and timely reflected by a run-time model of a higher abstraction level?

As demonstrated for the business processes domain, this can be achieved by introducing an intermediate computer-oriented run-time model and two separate lifecycles: one that (*timely*) abstracts the low level information into changes in the computer-oriented run-time model and one that interprets this information into a human-oriented run-time model, thus, completing the *causal* link. This general approach could potentially be adopted for other models@run.time use cases that have similar characteristics (big abstraction gap + uncontrolled deviations as described in Section 3.1). Additionally, to transform a run-time state model language into an evolution model language, extensions were proposed which effectively add a time dimension to relations and thus enable the realisation of temporal relations.

7.1.2 Main Contributions

Identification of BP run-time characteristics and composition of a holistic DBPMRT specification

One of the main objectives of this thesis was the identification of run-time characteristics of BPs and the design of a DBPMRT specification able to capture the holistic and descriptive reflection of a BPMS. The identified characteristics as well as a reference specification for a DBPMRT have been described in Chapter 4: a DBPMRT comprises information of all important type level perspectives of a business process, i.e. control-flow, resources, and performance, and is able to represent different levels of change, i.e. dynamism, variability, and reflectivity. According to these requirements reference meta-models which specify DBPMRT languages for state or evolution models, respectively, have been provided. They have been realised with the Eclipse Modelling Framework (EMF) using only the existent basic capabilities of EMF, i.e. special relations of models at run-time such as *temporal relations* were successfully substituted by artefacts modelled with standard EMF. Additionally, in order to deal with the challenge of differing abstraction levels (low level event data vs. high level BP model) it was proposed to distinguish between two different general types of run-time models: *computer-oriented* footprint models vs. *human-oriented* BP models. This conceptual differentiation is part of the main concept to maintain a causal connection in a run-time environment and described in Chapter 5.

Key Algorithms, Methodologies, and Frameworks

The following list of contributions are the outcome of addressing the remaining two objectives of this thesis (Chapters 5 and 6):

- A novel algorithm to establish a causal link between event data and a descriptive BP control-flow model: The *Constructs Competition Miner* (CCM) follows a top-down approach to directly discover block-structured process models which consist

of common BP-domain constructs and represent the main behaviour of the process. The algorithm was designed so that it can deal with exceptional and contradictory behaviour. This was achieved by letting the different supported constructs compete with each other for the most suitable solution from top to bottom using "soft" constraints and behaviour approximations based on global relations between activities abstracted from the event log.

- Modifications of the statically operating CCM algorithm in order to work in a real-time setting to detect changes in the BP control-flow while processing a stream of BP execution events: the *Dynamic Constructs Competition Miner* (DCCM). The DCCM applies the core principle of dividing the discovery methodology into two separate lifecycles with the Dynamic Footprint at its centre. An additional evaluation of the DCCM on the eHealth use case "DrugFusion" is provided in [182] (not part of the thesis).

- A dynamic concept for the event-based update of higher abstraction levels of the model (e.g. change in performance, control-flow, etc.) including concepts like "time-dependent ageing" and "discrete ageing". Additionally to the core ageing concept, modifications were provided which have been proven to successfully shorten the "warm-up" phase.

- A smart and time-efficient algorithm that tracks the fine-grained instance state of an enterprise system based on bit operations.

- A generic framework for Descriptive Business Process Models at Run-time, enabling (1) automated monitoring of an enterprise system by capturing the state of the system in a descriptive run-time model and (2) real-time reasoning based on the descriptive run-time model.

- An architecture that enables real-time reasoning based on DBPMRTs: The reasoning system can be considered as an autonomous MRT system that operates based on continuously updating model information. The solution proposed to address this challenge is a repository architecture with the DBPMRT at the centre providing synchronised access for the update and analysis agents.

- A methodology for predictive reasoning utilising BP simulations and integrating an external performance processing component: The (evolution) DBPMRT comprises current and historical data about all important perspectives on the type and instance level which can be used for short term simulations predicting future event sequences. The thesis proposes a simulation-based methodology that makes use of an external performance processing tool to process the simulated future events in the same way as the live events. As a result, not only historic and current performance values can be computed but also the predicted future development of these. This concept is also applicable for other event processing analyses, e.g. detection

of future bottlenecks, what-if analysis, which can be (re-)used to enable *proactive* decision support on future states of the system.

Implementation of Key Algorithms, Methodologies, and Frameworks

All presented algorithms, methodologies, and frameworks have been implemented as described in the thesis. Some parts have also also incorporated into other solutions, e.g. the performance prediction framework, in which case additional adapters or interfaces were implemented. Furthermore, multiple visualisation and demonstration tools were developed on the basis of the thesis' contributions[1] as local, mobile, or web applications. Three examples of these are shown in Annex C.

Exploitation

The contributions of this thesis were utilised and put into practice by SAP projects such as *Operational Process Intelligence* (OPInt), a project about monitoring the performance of large enterprise processes [206], *bizInsight*, an internal innovation project in the area of business intelligence which combines Business Dynamics, Process Discovery, and Process Simulation techniques to monitor and predict KPIs and PPIs. Also, main components and algorithms such as the DPDF and DCCM are integral parts of the digital preservation and risk management platform developed within the EU funded project TIMBUS [84]. Much of the work presented in this thesis was demonstrated at different internal or external events/locations, e.g. Lancaster University, Ulster University, Queen's University Belfast, DKOM Karlsruhe, or at scientific conferences. Furthermore, the main contributions have been published and presented at conferences (see Annex A).

7.2 Future Research Agenda

Improvements to the DBPMRT

- It could be beneficial for the run-time reasoning if the DBPMRT would reflect the system in a less general but more detailed way. One possible direction for future research is therefore to investigate how to make the DBPMRT more expressive and representative. This includes efforts to extend the specification by the ability to capture, for instance, (1) the system-specific data perspective, (2) more specialised control-flow elements such as OR-Split and OR-join, and/or (3) enable a more realistic representation of the organisational perspective - the current representation regards human resources as CPU-like processor while, in fact, humans are more complex entities: they can be sick or on holidays as well as have a varying efficiency

[1] Note, that the implementation of the demonstration tools was mainly carried out by undergraduate students under the supervision of the author.

for carrying out tasks or work on other, not recorded tasks that reduce their availability.

- The temporal relation for the evolution model was realised via extending the state model by substitute modelling artefacts which are not optimised for this type of relation. A more efficient way would be to enable the model expert to directly model "temporal relations" which are then automatically realised by using more efficient data structures that optimise access and update operations, e.g. similar to Google's *BigTable*[2]. Another option would be to investigate the possibility of specifying the DBPMRT as declarative language for which it might be easier to integrate the time dimension. Thus, a possible direction for future research is the analysis of these and similar other methods to effectively realise temporal relations.

Discovery Improvements

- The proposed algorithm (CCM) for establishing a causal link (only control-flow) has shown weaknesses in terms of discovering some of the more special constructs such as a loop over a parallel split (see Section 5.8.1). This construct is not always correctly identified with the current constraints. One possible direction for future research is the establishment of better suiting constraints which would likely improve the results of the CCM (as well as DCCM, DPDF, and higher-level reasoning). Furthermore, it should be investigated how additional BP constructs can be identified, e.g. OR-Split and -Join.

- Currently, if no sub-footprint is available for a set of activities, the footprint interpreter does not further analyse this set but creates a single representative activity. This behaviour makes the reasoning on such a model more complicated or even impossible since it may not be interpretable by the reasoning component. Future research should focus on mitigating this undesirable effect, e.g. through approximations or the use of the direct neighbours relation in order to retrieve a "close enough" control-flow for the activity subset. Such an alternative way would ensure that always a complete and analysable DBPMRT is available.

- Although a change is quickly detected, it currently takes an undesirable large amount of events/traces until the new process is appropriately reflected by the DBPMRT (see Section 5.8.2). This behaviour originates from the fact that the dynamic footprint and the interpreted business process are in a sort of intermediate state for a while until the influence of the old version of the business process becomes irrelevant in the footprint. This behaviour is not observable for the warm-up phase where the model is in most cases already correctly represented after a significantly shorter amount of time. This is an indication that resetting the dynamic footprint (or parts

[2] https://cloud.google.com/bigtable/ (accessed 21.12.2015)

of it) after detecting a change (which has been proven to happen almost instantly) could be a possible solution that should be investigated to address this problem. Other solutions might be of benefit to reduce the transition time and are be part of future research.

- Currently the trace influence factor t_{if} is a pre-specified value but in reality it is dependent on how many traces are needed to represent the complete behaviour of the model, i.e. if many possible instance variants exist then it takes many traces until this is represented in the footprint; if only a few variants exist the complete behaviour is quickly represented in the footprint. The complete behaviour is in turn strongly dependent on the number of activities in the model, since more activities usually mean more control-flow behaviour. Hence, a possible future research direction to improve the autonomy of the solution is to investigate ways to make the trace influence factor adapt dynamically, e.g. by making it dependent on the number of observed activities. This step could also shorten the transition time (see previous point).

Further Efforts to Promote Scalability

The division of the discovery method into event processing and footprint interpretation transferred the run-time cost-expensive interpretation into a lifecycle separate from the event lifecycle. This allows for a parallel and arbitrarily execution on demand or in a re-occurring fashion. Since the processing is featuring a constant run-time for each event that occurs (only dependant on the amount of activities) the overall processing demand has a linear run-time in terms of the overall number of occurring events, i.e. the required computation power increases with an increasing event frequency. This should be further mitigated through techniques used in the big data domain. For instance, each machine could be responsible for the processing and creation of specified instance footprints. Following this approach only the eventual update of the overall dynamic footprint needs to be centralised. With this the algorithmic run-time of the event processing becomes linear in terms of instance frequency instead of event frequency (the former usually occurs significantly more seldom). One direction for future research is the investigation of scalability approaches like the above to further improve the algorithmic run-time of the DPDF (and DCCM).

Adaptive Reasoning and System Modification

The work presented in this thesis is a major step towards shortening the BPM lifecycle. In order to close the loop and fully automate the management of BPs, it is required to employ adaptive reasoning and modification techniques. This is not easily achievable since it involves a new set of challenges, e.g. modification policies, change validation, etc. In the area of workflows (fully automated BPs) these challenges have been addressed,

however, always under the assumption that no uncontrolled changes are introduced from outside of the system. That means the supervising system records and/or controls all type level changes directly which essentially eliminates the abstraction gap and thus the need for (dynamic) process discovery. One possible future direction is therefore to investigate whether the lessons in the area of adaptive reasoning in workflow management can also be fully or partly applied to the business process management domain.

Investigation of other Domains

The contributions of this thesis might also apply in other domains which have similar characteristics (big abstraction gap + uncontrolled deviations) than the BP domain for which reasoning on the current run-time state of the system might be beneficial, e.g. risk management. Thus, investigating these domains for their potential role of run-time models is a possible direction for future research and might lead to further contributions in the domain of DBPMRT or MRT and BPM in general.

7.3 Final Remarks

This thesis has investigated the potential of adopting and enhancing principles and mechanisms from the models@run.time domain to the business process domain for the purpose of run-time reasoning. The proposed meta-model artefacts, algorithms, methodologies, and framework concepts are a result of this investigation. By taking the steps proposed in this thesis general challenges of business process management, e.g. dealing with frequently changing processes and shortening the business process life cycle, have been addressed. This also contributed to research in models@run.time by providing a complex real-world use case and a principled approach for dealing with volatile models@run.time of a higher abstraction level.

The author hopes that this thesis will contribute to a more dynamic handling of business processes in the future and may serve as a reference for other domains where the reasoning on descriptive run-time time models may be beneficial.

Appendix A

Publications

For overview purposes this appendix contains a collection of all research contributions made by the author during the course of the PhD studies. A mapping of the publications to the contributions presented in this thesis is provided in Section 1.7. The publications listed are chronologically sorted by date of publication (which is not necessarily the same as the date of the associated conference/workshop).

Conference and Workshop Papers as Principal Author

2015

1. ***Dynamic Constructs Competition Miner - Occurrence- vs. Time-based Ageing***, **David Redlich**, Thomas Molka, Wasif Gilani, Gordon S. Blair, Awais Rashid. In Post-Proceedings of the 4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2014), volume 237 of Lecture Notes in Business Information Processing, pages 79-106. Springer.

2. ***Evaluation of the Dynamic Construct Competition Miner for an eHealth System***, **David Redlich**, Mykola Galushka, Thomas Molka, Wasif Gilani, Gordon S. Blair, Awais Rashid. In Business Information Systems - 18th International Conference, BIS 2015, Proceedings, volume 208 of Lecture Notes in Business Information Processing, pages 115–126. Springer.

2014

3. ***Scalable Dynamic Business Process Discovery with the Constructs Competition Miner***,
David Redlich, Thomas Molka, Wasif Gilani, Gordon S. Blair, Awais Rashid. In Proceedings of the 4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2014), volume 1293 of CEUR Workshop Proceedings, pages 91–107. CEUR-WS.org.

4. ***Introducing a Framework for Scalable Dynamic Process Discovery***,
   David Redlich, Wasif Gilani, Thomas Molka, Marc Drobek, Awais Rashid, Gordon S. Blair. In Advances in Enterprise Engineering VIII - 4th Enterprise Engineering Working Conference, EEWC 2014. Proceedings, volume 174 of Lecture Notes in Business Information Processing, pages 151–166. Springer.

5. ***Constructs Competition Miner: Process Control-flow Discovery of BP-domain Constructs***,
   David Redlich, Thomas Molka, Wasif Gilani, Gordon S. Blair, Awais Rashid. In Business Process Management - 12th International Conference, BPM 2014. Proceedings, volume 8659 of Lecture Notes in Computer Science, pages 134–150. Springer.

6. ***Model-driven Engineering in Practice: Integrated Performance Decision Support for Process-centric Business Impact Analysis***,
   David Redlich, Ulrich Winkler, Thomas Molka, Wasif Gilani. In ACM/SPEC International Conference on Performance Engineering, ICPE 2014, pages 247–258. ACM.

7. ***Research Challenges for Business Process Models at Run-time***,
   David Redlich, Gordon S. Blair, Awais Rashid, Thomas Molka, Wasif Gilani. In Models@run.time - Foundations, Applications, and Roadmaps [Dagstuhl Seminar 11481, 2011], volume 8378 of Lecture Notes in Computer Science, pages 208–236. Springer.

2012

8. ***MDE in Practice: Process-centric Performance Prediction via Simulation in Real-time***,
   David Redlich, Stefanie Platz, Wasif Gilani. In Joint Proceedings of co-located Events at the 8th European Conference on Modelling Foundations and Applications (ECMFA) - Tool Presentations, pages 336–339.

2011

9. ***Event-driven Process-centric Performance Prediction via Simulation***,
   David Redlich, Wasif Gilani. In Business Process Management Workshops - BPM 2011 International Workshops, Revised Selected Papers, Part I, volume 99 of Lecture Notes in Business Information Processing, pages 473–478. Springer.

For all listed publications the author was the main responsible person in terms of the involved research activities as well as the authorship of the publication. With the exception of publication 6 all can be directly mapped to contributions presented in this thesis (see Section 1.7).

Conference and Workshop Papers as Contributing Author

2015

10. ***Diversity Guided Evolutionary Mining of Hierarchical Process Models***,
Thomas Molka, **David Redlich**, Marc Drobek, Xiao-Jun Zeng, Wasif Gilani. In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, pages 1247–1254. ACM.

11. ***Evolutionary Computation Based Discovery of Hierarchical Business Process Models***,
Thomas Molka, **David Redlich**, Wasif Gilani, Xiao-Jun Zeng, Marc Drobek. In Business Information Systems - 18th International Conference, BIS 2015, Proceedings, volume 208 of Lecture Notes in Business Information Processing, pages 191–204. Springer.

2014

12. ***Advanced Business Simulations - Incorporating Business and Process Execution Data***,
Marc Drobek, Wasif Gilani, **David Redlich**, Thomas Molka, Danielle Soban. In Business Modeling and Software Design - 4th International Symposium, BMSD 2014, Revised Selected Papers, volume 220 of Lecture Notes in Business Information Processing, pages 119–137. Springer.

13. ***A Process-oriented Performance and Risk Management Workbench***,
Wasif Gilani, Mykola Galushka, Thomas Molka, **David Redlich**, Ying Du, Marc Drobek. In eChallenges e-2014, 2014 Conference, pages 1–9.

14. ***Conformance Checking for BPMN-based Process Models***,
Thomas Molka, **David Redlich**, Marc Drobek, Artur Caetano, Xiao-Jun Zeng, Wasif Gilani. In Symposium on Applied Computing, SAC 2014, pages 1406–1413. ACM.

15. ***Mechanisms for Leveraging Models at Runtime in Self-adaptive Software***,
Amel Bennaceur, Robert B. France, Giordano Tamburrelli, Thomas Vogel, Pieter J. Mosterman, Walter Cazzola, Fábio M. Costa, Alfonso Pierantonio, Matthias Tichy, Mehmet Aksit, Pär Emmanuelson, Gang Huang, Nikolaos Georgantas, **David Redlich**. In Models@run.time - Foundations, Applications, and Roadmaps[Dagstuhl Seminar 11481, 2011], volume 8378 of Lecture Notes in Computer Science, pages 19–46. Springer.

2013

16. ***TIMBUS: Digital Preservation for Timeless Business Processes and Services***,
Wasif Gilani, **David Redlich**, Mykola Galushka, Thomas Molka, Ying Du. In eChallenges e-2013, 2013 Conference.

2010

17. ***Combination of a Discrete Event Simulation and an Analytical Performance Analysis through Model-Transformations***,
Tomasz Porzucek, Stephan Kluth, Mathias Fritzsche, **David Redlich**. In 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS 2010, pages 183–192. IEEE Computer Society

For the listed publications the author contributed to the involved research activities as well as the co-authorship of the publication. While some of the thesis' concepts are integral parts of the publications 12, 13, and 16, the research carried out for the other publications is related to the thesis' topic but not within the scope of the presented concepts.

Patents

18. ***A Novel Closed Loop Business Dynamics Solution***,
Marc Drobek, Wasif Gilani, **David Redlich**, Thomas Molka. US Patent, Reference number: 141165US01; Application number: US 14/716,161 (filed in 2015)

19. ***Model-based Business Continuity Management***,
Ulrich Winkler, Wasif Gilani, **David Redlich**. US Patent 8,457,996. (2013)

Both patents yield contributions in the field of model-based simulations for business processes.

Public Project Deliverables

The thesis' concepts are part of the contributions of the EU funded project TIMBUS. The author was either main or co-author of the following deliverables:

- D6.10: Refinements to Populating and Accessing Context Model (2014)
- D6.5: Populating and Accessing the Context Model (2013)
- D8.2: Use Case Specific Risks - Civil Engineering Infrastructures (2012)
- D9.2: Use Case Specific Risks - eScience and Mathematical Simulations (2012)
- D4.2: Dependency Models: Definition of a Formalism to Express Dependencies and Relations between Technological, Business and Organizational Components and Processes (2012)

Appendix B

Example Footprint Update with Ageing

Given that DFP_{34} is the following (equivalent of example footprint $FP^{L_1,A}_{\{\},\{\}}$ from page 144):

$$\begin{array}{c} \\ Oon(x): \\ Oov(x): \\ Fel(x): \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{pmatrix} 0.56 & 0.56 & 0.18 & 0.44 & 0.44 & 0.38 & 0.35 & 0.44 \\ 0.91 & 0.91 & 0.35 & 0.62 & 0.62 & 1.09 & 0.65 & 0.85 \\ 0.15 & 0.41 & 0 & 0 & 0 & 0.32 & 0.12 & 0 \end{pmatrix} \end{array}$$

$x \rhd \rhd y$:

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{pmatrix} 0.18 & 0.32 & 0.18 & 0.44 & 0.44 & 0 & 0 & 0 \\ 0.41 & 0.18 & 0.18 & 0.44 & 0.44 & 0 & 0 & 0 \\ 0.18 & 0.18 & 0.18 & 0.18 & 0.18 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.18 & 0.32 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.29 & 0.18 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.24 & 0.24 & 0.32 \\ 0 & 0 & 0 & 0 & 0 & 0.29 & 0.29 & 0.29 \\ 0 & 0 & 0 & 0 & 0 & 0.29 & 0.24 & 0.24 \end{pmatrix} \end{array}$$

$x \rhd y$:

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{pmatrix} 0 & 0.15 & 0.18 & 0.44 & 0.44 & 0 & 0 & 0 \\ 0.41 & 0 & 0.18 & 0.44 & 0.44 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.18 & 0.18 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.32 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.24 & 0.32 \\ 0 & 0 & 0 & 0 & 0 & 0.06 & 0 & 0.06 \\ 0 & 0 & 0 & 0 & 0 & 0.06 & 0.24 & 0 \end{pmatrix} \end{array}$$

And the trace footprint TFP_{35} for the new trace $[b, a, c, a, b, c, b, a, d, e, e, d]$ is:

$$\begin{array}{c} \\ Oon(x): \\ Oov(x): \\ Fel(x): \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 3 & 3 & 2 & 2 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$x \rhd \rhd y$:

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$x \rhd y$:

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Then for trace influence factor $t_{if} = 0.1$ the new dynamic overall footprint DFP_{35} is calculated by

$$DFP_{35} = 0.1 * TFP_{35} + 0.9 * DFP_{34}$$

That means that all vectors and matrices are multiplied with their respective factor (0.1 or 0.9) and added, e.g. for the $x \rhd\rhd y$ relation:

$$(x \rhd\rhd y)_{35} = 0.1 * \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} + 0.9 * \begin{pmatrix} 0.18 & 0.32 & 0.18 & 0.44 & 0.44 & 0 & 0 & 0 \\ 0.41 & 0.18 & 0.18 & 0.44 & 0.44 & 0 & 0 & 0 \\ 0.18 & 0.18 & 0.18 & 0.18 & 0.18 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.18 & 0.32 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.29 & 0.18 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.24 & 0.24 & 0.32 \\ 0 & 0 & 0 & 0 & 0 & 0.29 & 0.29 & 0.29 \\ 0 & 0 & 0 & 0 & 0 & 0.29 & 0.24 & 0.24 \end{pmatrix}$$

$$\Rightarrow \quad (x \rhd\rhd y)_{35} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{pmatrix} 0.26 & 0.39 & 0.26 & 0.50 & 0.50 & 0 & 0 & 0 \\ 0.47 & 0.26 & 0.26 & 0.50 & 0.50 & 0 & 0 & 0 \\ 0.26 & 0.26 & 0.26 & 0.26 & 0.26 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.26 & 0.39 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.36 & 0.26 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.22 & 0.22 & 0.29 \\ 0 & 0 & 0 & 0 & 0 & 0.26 & 0.26 & 0.26 \\ 0 & 0 & 0 & 0 & 0 & 0.26 & 0.22 & 0.22 \end{pmatrix} \end{array}$$

This operation is applied to each respective vector and matrix. The resulting "aged" overall footprint DFP_{35} is the following:

$$\begin{array}{c} \\ Oon(x): \\ Oov(x): \\ Fel(x): \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{pmatrix} 0.60 & 0.60 & 0.26 & 0.50 & 0.50 & 0.34 & 0.32 & 0.40 \\ 1.12 & 1.12 & 0.52 & 0.76 & 0.76 & 0.98 & 0.59 & 0.77 \\ 0.14 & 0.47 & 0 & 0 & 0 & 0.29 & 0.11 & 0 \end{pmatrix} \end{array}$$

$x \rhd\rhd y$:

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{pmatrix} 0.26 & 0.39 & 0.26 & 0.50 & 0.50 & 0 & 0 & 0 \\ 0.47 & 0.26 & 0.26 & 0.50 & 0.50 & 0 & 0 & 0 \\ 0.26 & 0.26 & 0.26 & 0.26 & 0.26 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.26 & 0.39 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.36 & 0.26 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.22 & 0.22 & 0.29 \\ 0 & 0 & 0 & 0 & 0 & 0.26 & 0.26 & 0.26 \\ 0 & 0 & 0 & 0 & 0 & 0.26 & 0.22 & 0.22 \end{pmatrix} \end{array}$$

$x \rhd y$:

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{pmatrix} 0 & 0.14 & 0.26 & 0.50 & 0.50 & 0 & 0 & 0 \\ 0.47 & 0 & 0.26 & 0.50 & 0.50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.26 & 0.26 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.39 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.16 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.22 & 0.29 \\ 0 & 0 & 0 & 0 & 0 & 0.05 & 0 & 0.05 \\ 0 & 0 & 0 & 0 & 0 & 0.05 & 0.22 & 0 \end{pmatrix} \end{array}$$

Appendix C

Visualisation Tools

C.1 DBPMRT Visualisation

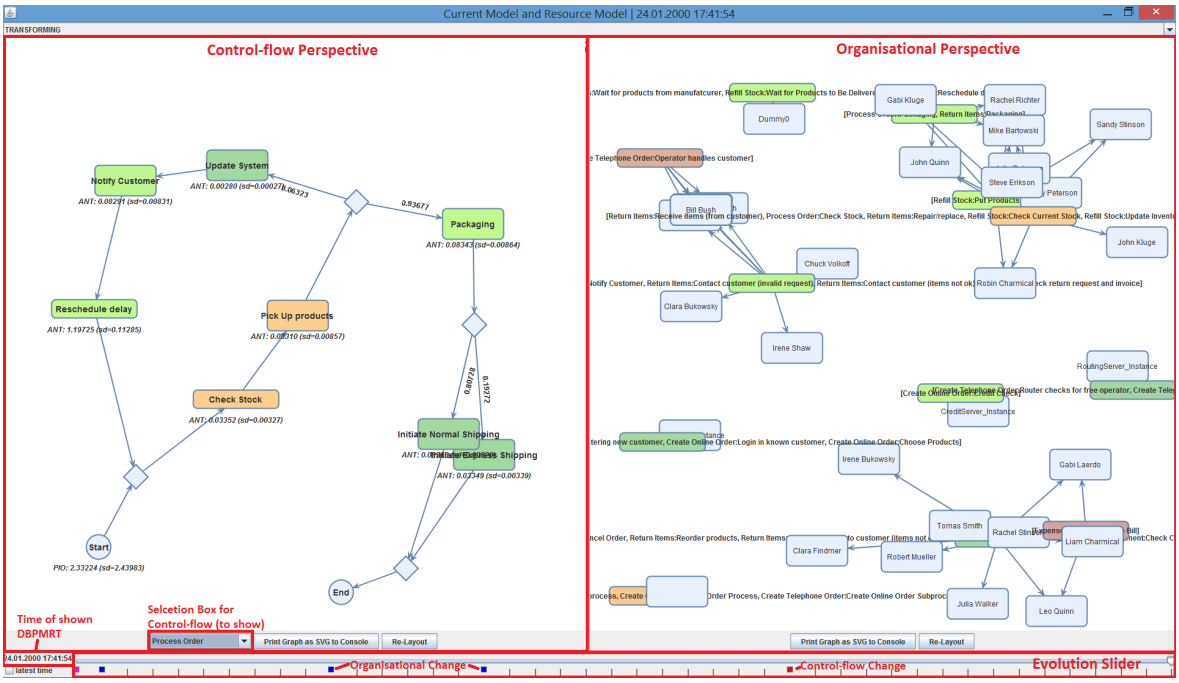


Fig. C.1 DBPMRT Evolution Visualisation (Element Descriptions annotated in red)

C.2 KPI/PPI Visualisations

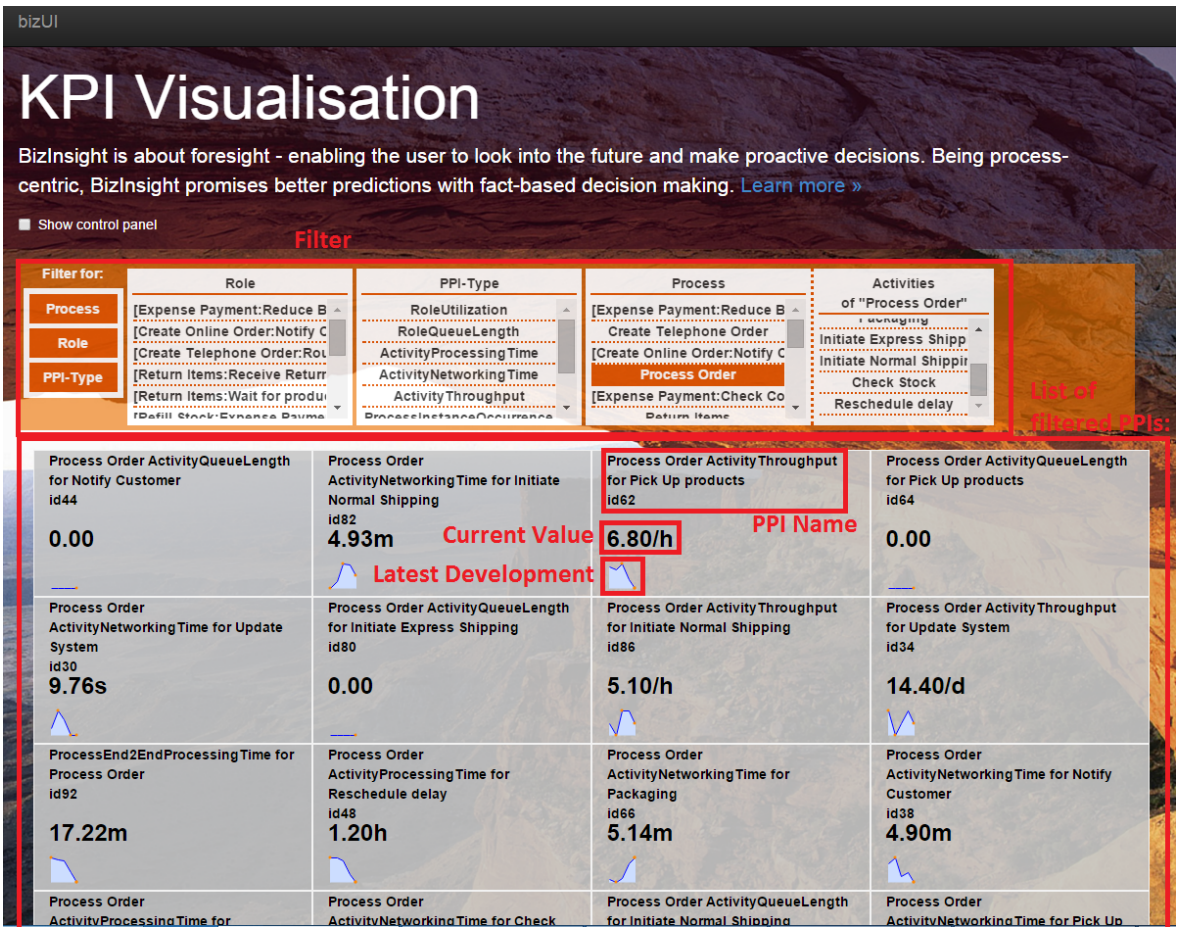


Fig. C.2 PPI Dashboard (Element Descriptions annotated in red)

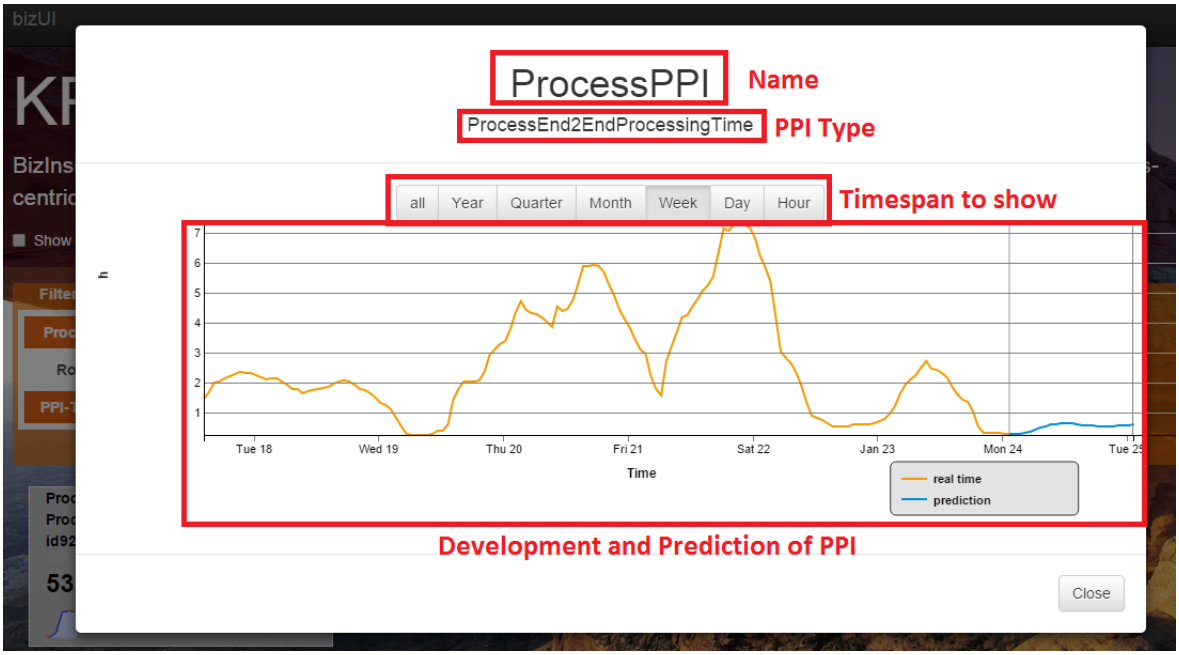


Fig. C.3 PPI Popup - when clicked on PPI in Dashboard (Descriptions annotated in red)

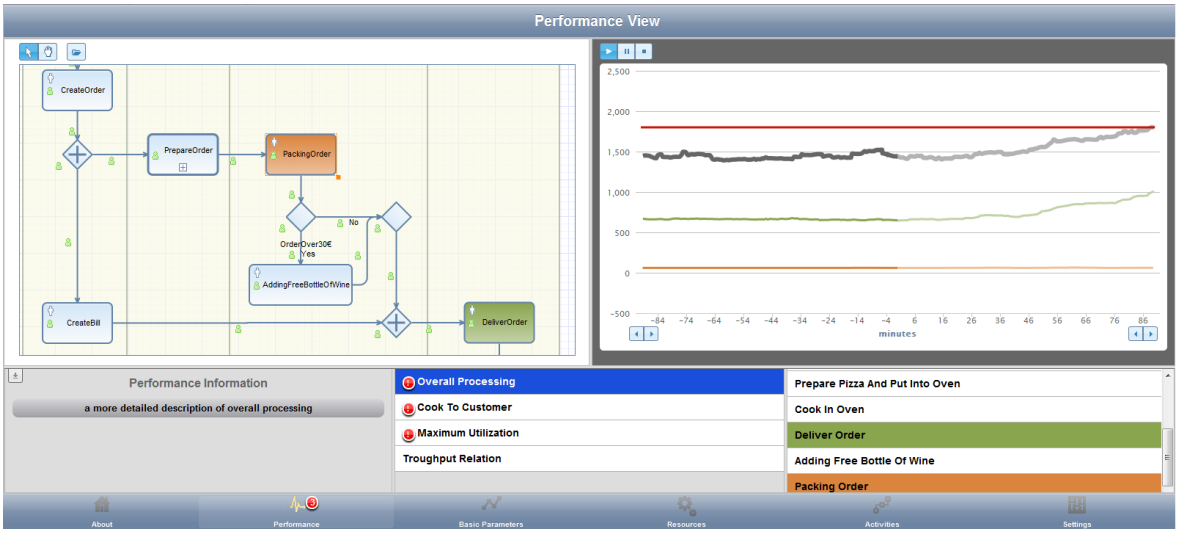


Fig. C.4 Tablet App for PPI Visualisation

Appendix D

Additional Processes in Case Study

D.1 Expense Payment

The "Expense Payment" BP is a sub-Process of the "Refill Stock" BP (see Figure D.2) but also independently initiated for weekly/monthly fixed payments.

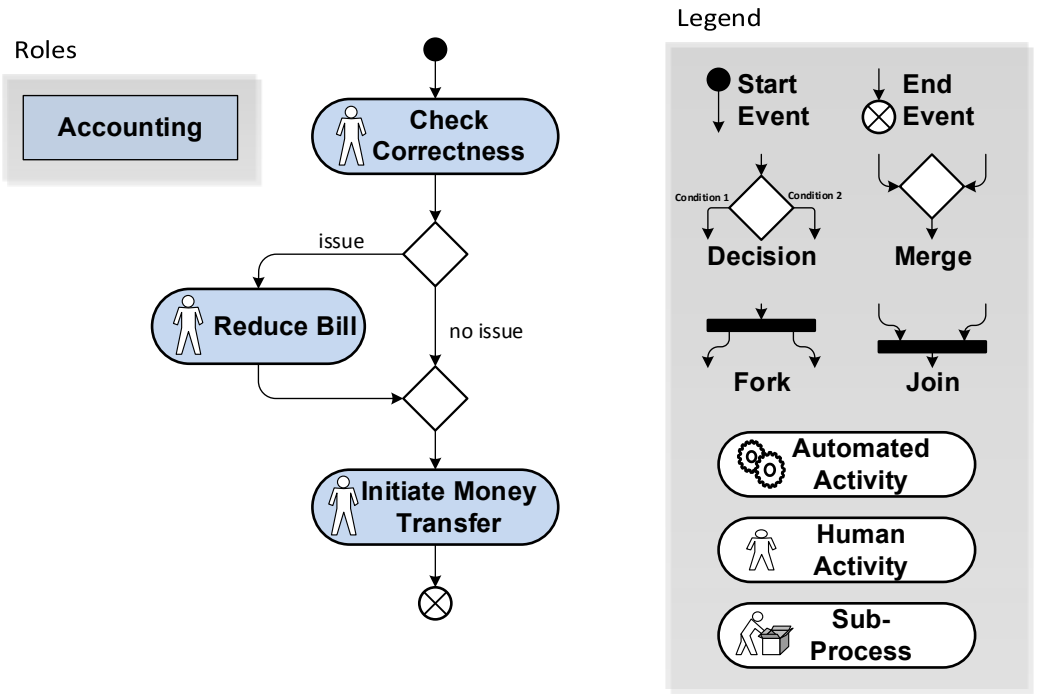


Fig. D.1 The Planned "Expense Payment" Business Process in the Akron Heating Company

D.2 Refill Stock

The "Refill Stock" BP is periodically initiated approximately every two days, checking and refilling the stock items to avoid delays in the delivery (when items are not in stock). The "Expense Payment" BP is a sub-Process of the "Refill Stock" BP.

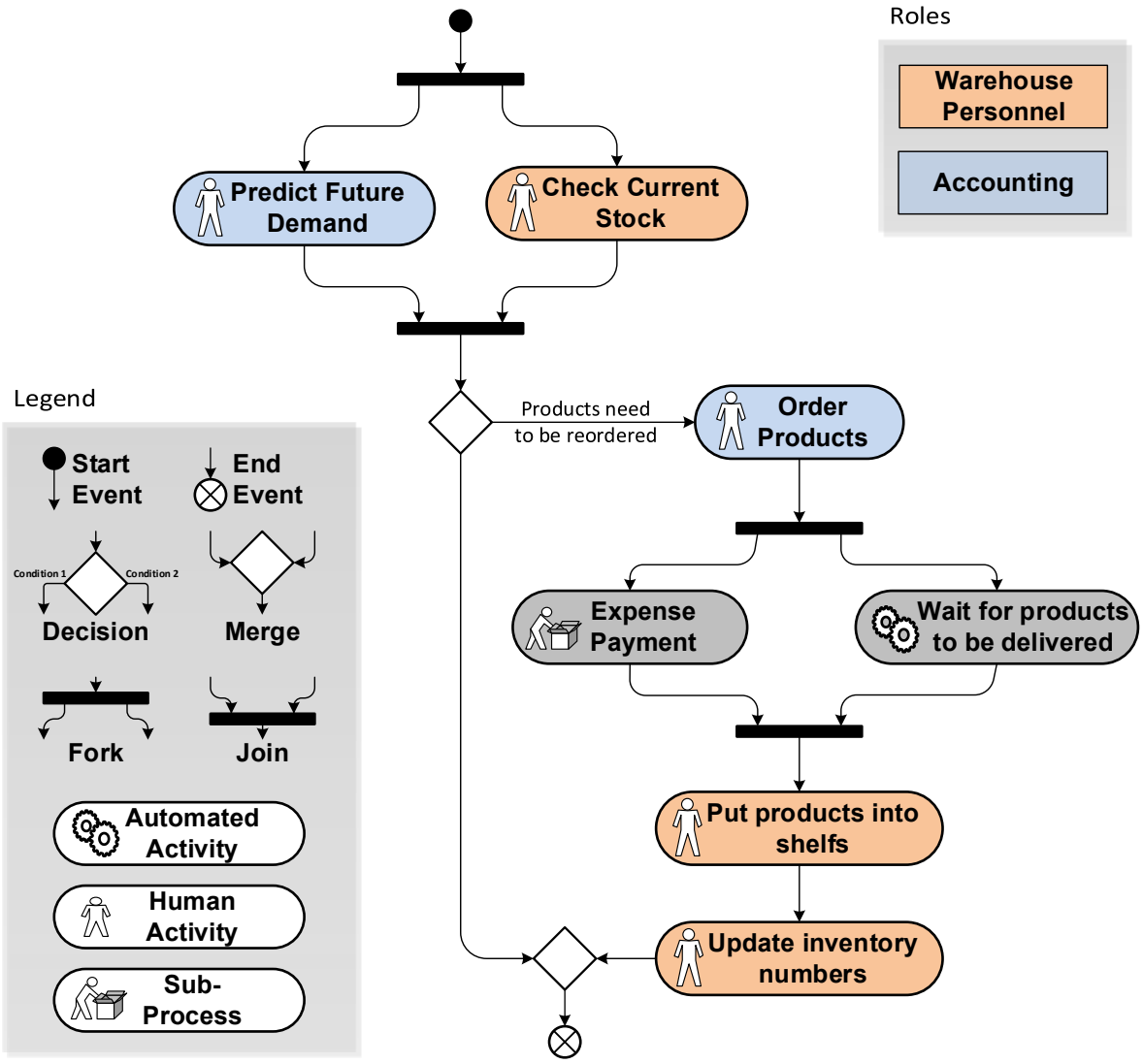


Fig. D.2 The Planned "Refill Stock" Business Process in the Akron Heating Company

D.3 Return Item

The "Return Item" BP is a complex process which is initiated when customers return their order or parts of them. In the Akron Heating case scenario this happens regularly with approximately 15% of the orders being sent back (in full or partly).

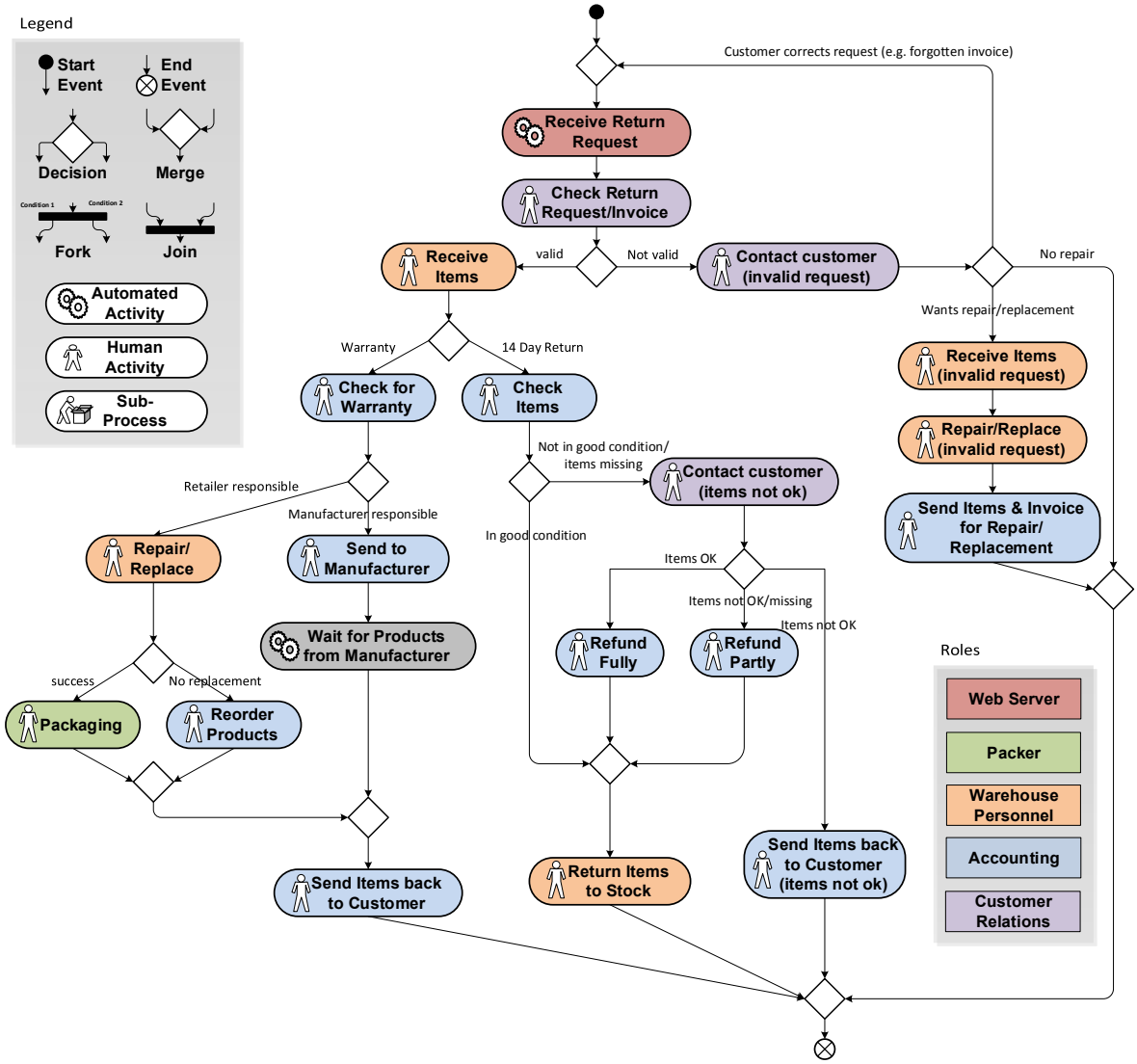


Fig. D.3 The Planned "Return Item" Business Process in the Akron Heating Company

Appendix E

Additional Evaluation Results for Case Study

E.1 Business Processes

E.1.1 Expense Payment

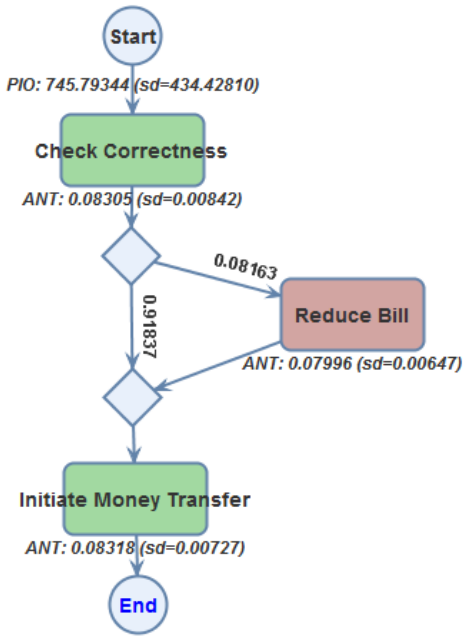


Fig. E.1 Discovered Control-flow of the *Expense Payment* BP

E.1.2 Refill Stock

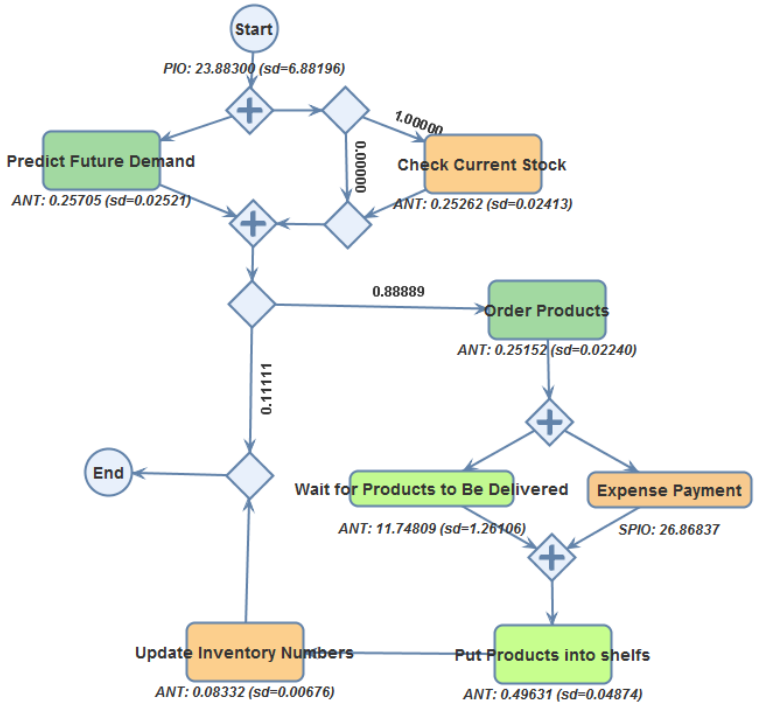


Fig. E.2 Discovered Control-flow of the *Refill Stock* BP

E.1.3 Return Item

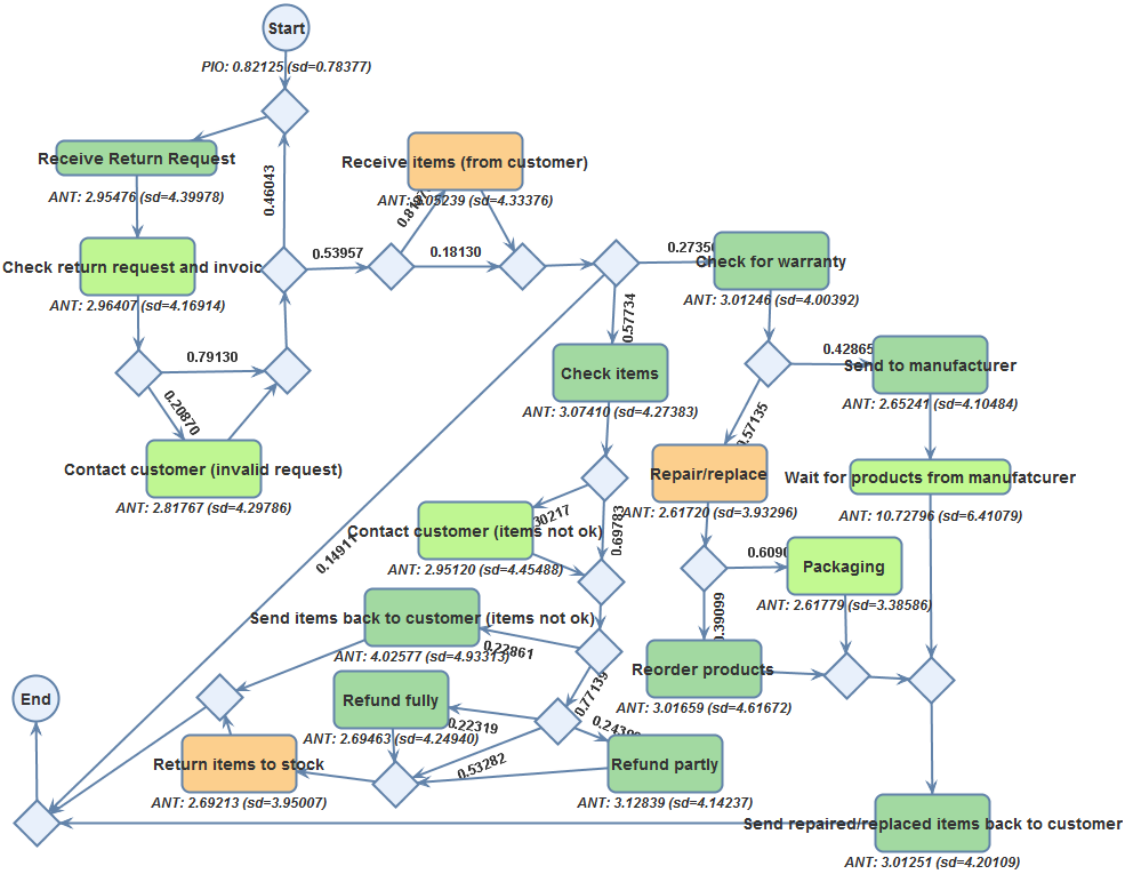


Fig. E.3 Discovered Control-flow of the *Return Item* BP

E.2 Performance

End-To-End Processing Time for Return Items BP

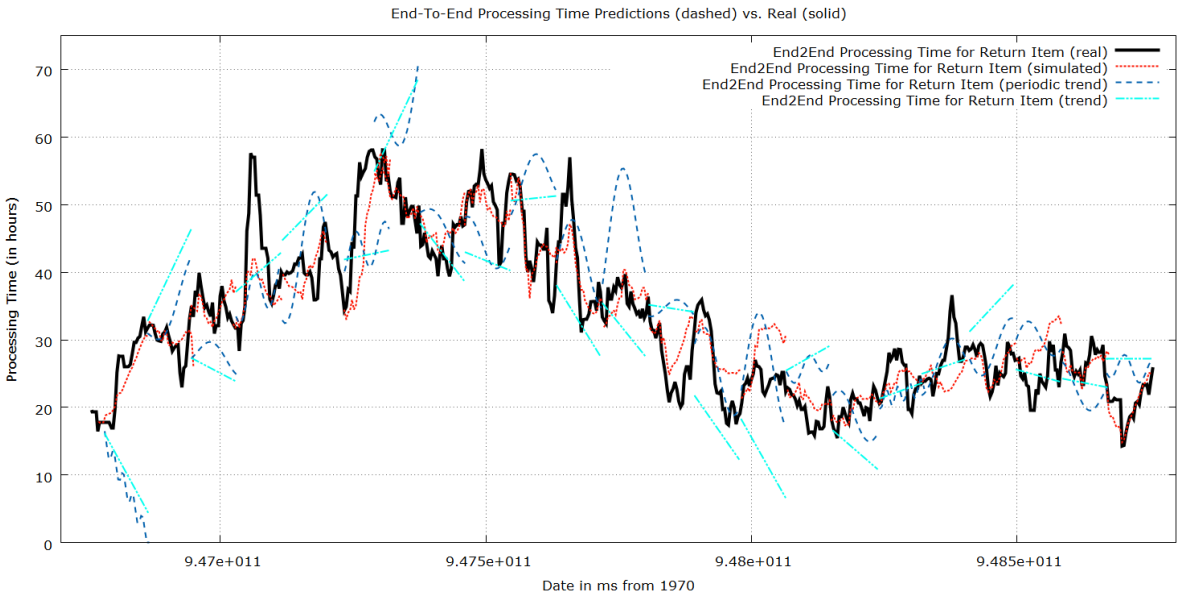


Fig. E.4 Prediction Results vs. Real Development for *End-to-End Processing Time* of "Return Items" BP

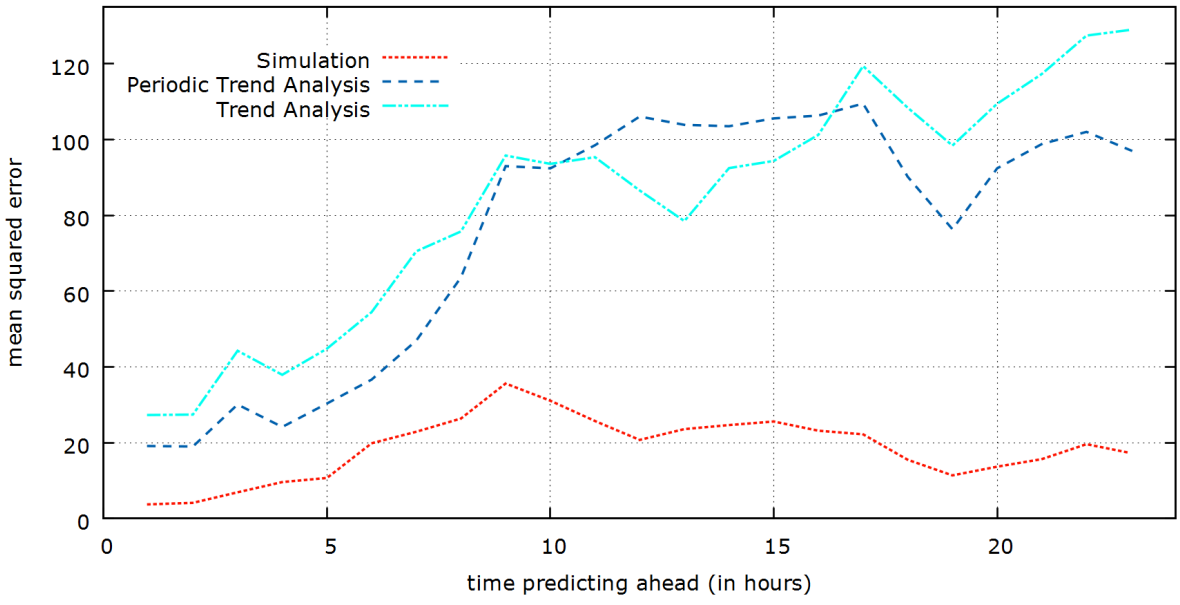


Fig. E.5 MSE of the Prediction Methods for *End-to-End Processing Time* of "Return Items" BP

Utilisation of Packer (Role)

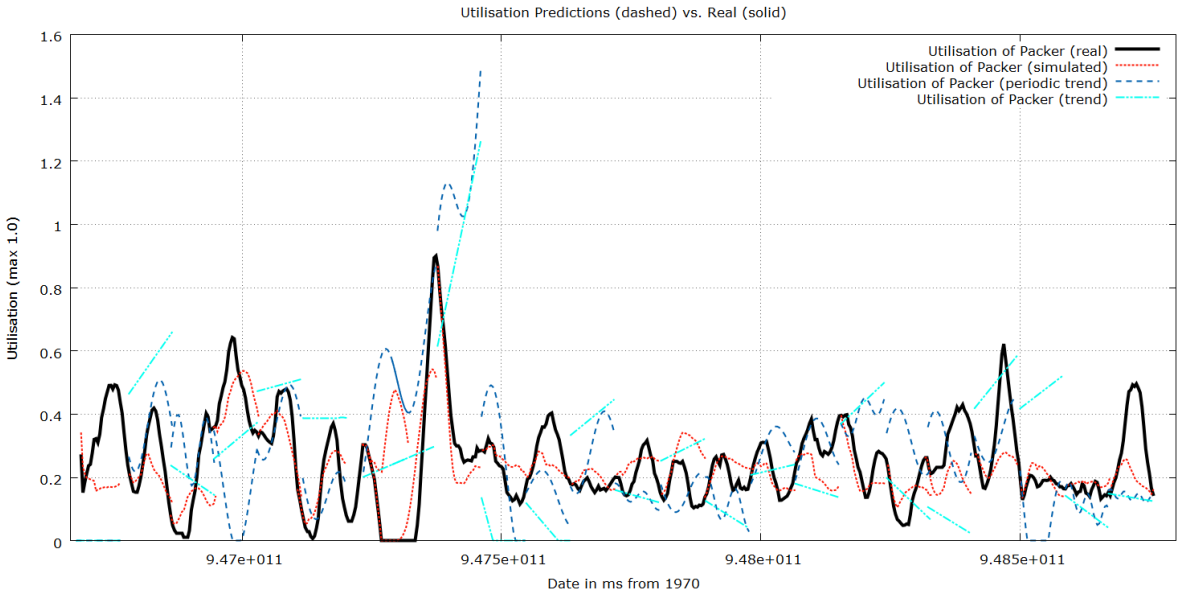


Fig. E.6 Prediction Results vs. Real Development for *Utilisation* of "Packer" Role

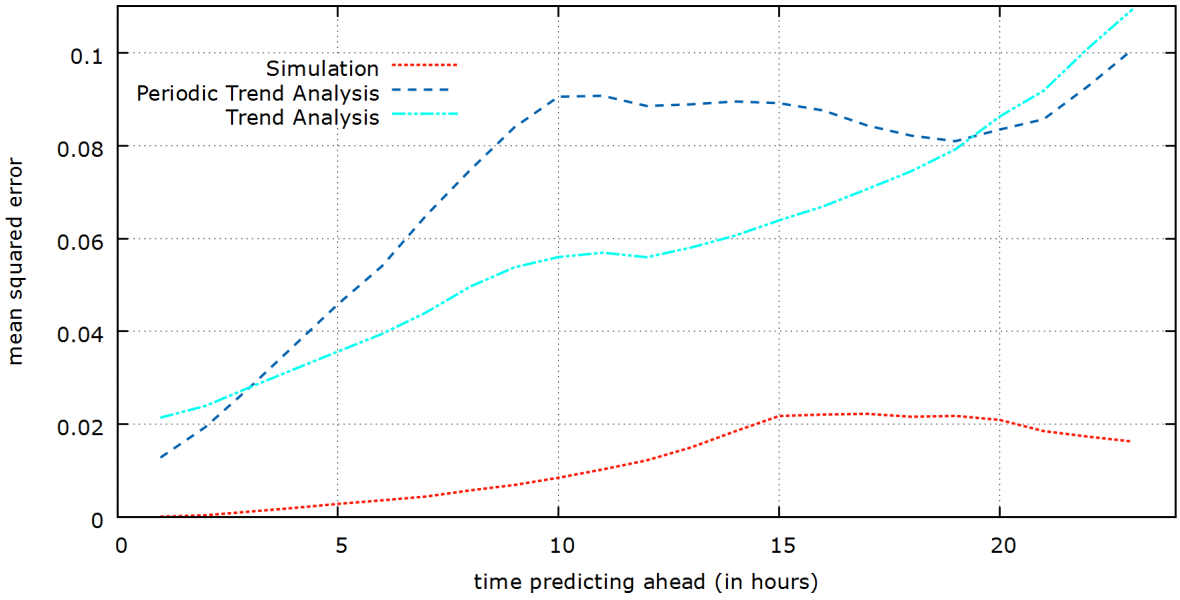


Fig. E.7 MSE of the Prediction Methods for *Utilisation* of "Packer" Role

Throughput of 'Initiate Express Shipping' Activity

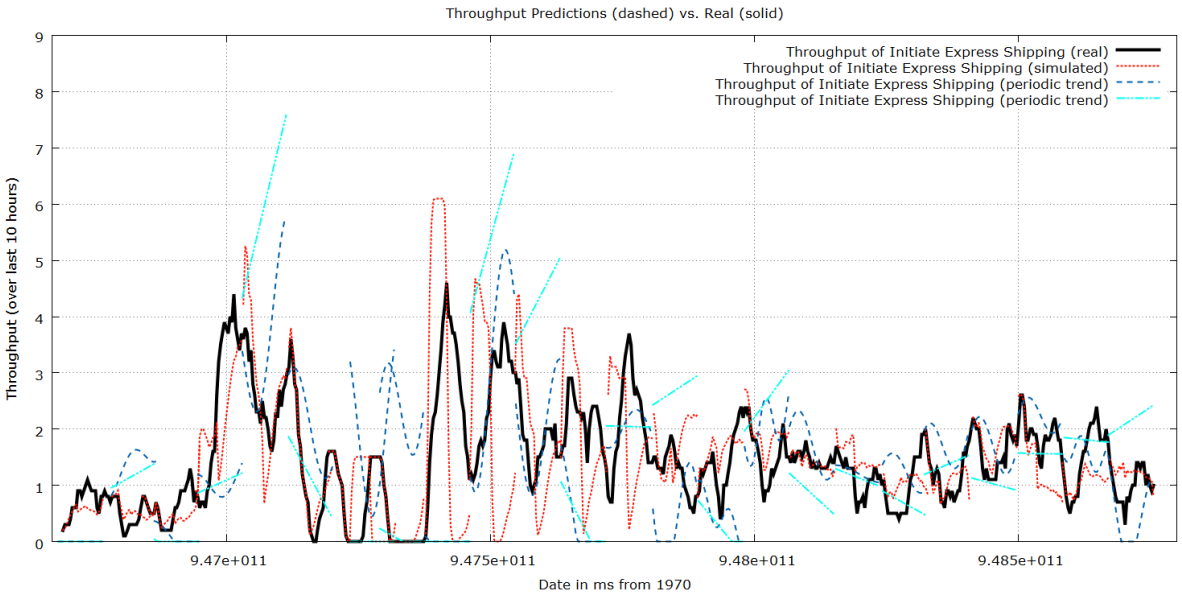


Fig. E.8 Prediction Results vs. Real Development for *Throughput* of "Initiate Express Shipping" Activity

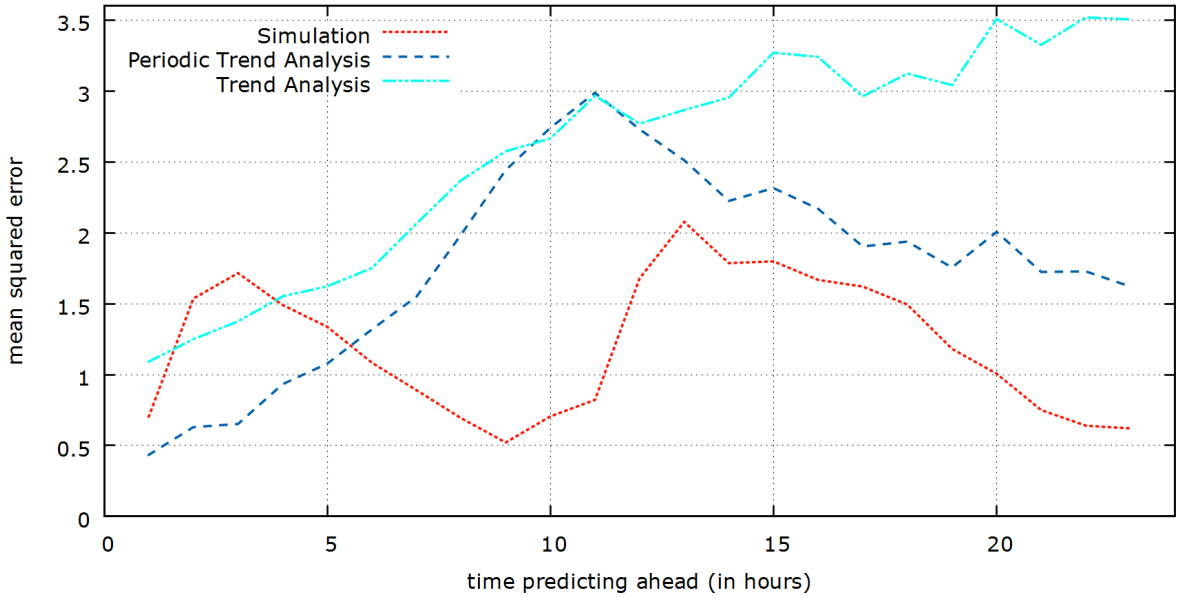


Fig. E.9 MSE of the Prediction Methods for *Throughput* of "Initiate Express Shipping" Activity

References

[1] Aagedal, J. Ø. (2001). *Quality of Service Support in Development of Distributed Systems*. PhD thesis, University of Oslo.

[2] Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B. F., and van der Aalst, W. M. P. (2012). Alignment based precision checking. In *Business Process Management Workshops - BPM 2012 International Workshops, Revised Papers*, volume 132 of *Lecture Notes in Business Information Processing*, pages 137–149. Springer.

[3] Adriansyah, A., van Dongen, B. F., and van der Aalst, W. M. P. (2011). Conformance checking using cost-based fitness analysis. In *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2011*, pages 55–64. IEEE Computer Society.

[4] Agrawal, R., Gunopulos, D., and Leymann, F. (1998). Mining process models from workflow logs. In *Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, Valencia, Spain, March 23-27, 1998, Proceedings*, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483. Springer.

[5] Allilaire, F., Bézivin, J., Jouault, F., and Kurtev, I. (2006). Atl – eclipse support for model transformation. In *IN: PROC. OF THE ECLIPSE TECHNOLOGY EXCHANGE WORKSHOP (ETX) AT ECOOP*.

[6] AnyLogic (2015). AnyLogic - multimethod simulation software. http://www.anylogic.com/ (accessed June 16th, 2015).

[7] Beltrame, G., Sciuto, D., Silvano, C., Lyonnard, D., and Pilkington, C. (2006). Exploiting TLM and object introspection for system-level simulation. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2006*, pages 100–105. European Design and Automation Association, Leuven, Belgium.

[8] Bencomo, N. (2008). *Supporting the Modelling and Generation of Reflective Middleware Families and Applications using Dynamic Variability*. PhD thesis, Computing Department, Lancaster University.

[9] Bencomo, N. (2009). On the use of software models during software execution. In *Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering*, MISE '09, pages 62–67, Washington, DC, USA. IEEE Computer Society.

[10] Bencomo, N., Blair, G. S., Götz, S., Morin, B., and Rumpe, B. (2013). Report on the 7th international workshop on models@run.time. *ACM SIGSOFT Software Engineering Notes*, 38(1):27–30.

[11] Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., and Letier, E. (2010). Requirements reflection: requirements as runtime entities. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2, ICSE 2010*, pages 199–202. ACM.

[12] Bennaceur, A., France, R. B., Tamburrelli, G., Vogel, T., Mosterman, P. J., Cazzola, W., Costa, F. M., Pierantonio, A., Tichy, M., Aksit, M., Emmanuelson, P., Huang, G., Georgantas, N., and Redlich, D. (2011). Mechanisms for leveraging models at runtime in self-adaptive software. In *Models@run.time - Foundations, Applications, and Roadmaps [Dagstuhl Seminar 11481, 2011].*, volume 8378 of *Lecture Notes in Computer Science*, pages 19–46. Springer.

[13] Bergenthum, R., Desel, J., Lorenz, R., and Mauser, S. (2007). Process mining based on regions of languages. In *Business Process Management, 5th International Conference, BPM 2007, Proceedings*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer.

[14] Bézivin, J. (2005). On the unification power of models. *Software and System Modeling*, 4(2):171–188.

[15] Bézivin, J., Jouault, F., and Valduriez, P. (2004). On the need for megamodels. In *Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*.

[16] Blair, G. S., Bencomo, N., and France, R. B. (2009). Models@ run.time. *IEEE Computer*, 42(10):22–27.

[17] Blair, G. S., Coulson, G., Clarke, M., and Parlavantzas, N. (2001). Performance and integrity in the openorb reflective middleware. In *Metalevel Architectures and Separation of Crosscutting Concerns, Third International Conference, REFLECTION 2001, Proceedings*, volume 2192 of *Lecture Notes in Computer Science*, pages 268–269. Springer.

[18] Blumendorf, M., Lehmann, G., Feuerstack, S., and Albayrak, S. (2008). Executable models for human-computer interaction. In *Interactive Systems. Design, Specification, and Verification, 15th International Workshop, DSV-IS 2008, Revised Papers*, volume 5136 of *Lecture Notes in Computer Science*, pages 238–251. Springer.

[19] Bodenstaff, L., Wombacher, A., Reichert, M., and Wieringa, R. (2010). Made4ic: an abstract method for managing model dependencies in inter-organizational cooperations. *Service Oriented Computing and Applications*, 4(3):203–228.

[20] Bose, R. J. C. (2012). *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics*. PhD thesis, Eindhoven University of Technology.

[21] Bose, R. P. J. C., van der Aalst, W. M. P., Zliobaite, I., and Pechenizkiy, M. (2011). Handling concept drift in process mining. In *Advanced Information Systems Engineering - 23rd International Conference, CAiSE 2011. Proceedings*, volume 6741 of *Lecture Notes in Computer Science*, pages 391–405. Springer.

[22] Bose, R. P. J. C., van der Aalst, W. M. P., Zliobaite, I., and Pechenizkiy, M. (2013). Dealing with concept drifts in process mining : a case study in a dutch municipality. Technical Report No. BPM-13-13.

[23] Bose, R. P. J. C., van der Aalst, W. M. P., Zliobaite, I., and Pechenizkiy, M. (2014). Dealing with concept drifts in process mining. *IEEE Trans. Neural Netw. Learning Syst.*, 25(1):154–171.

[24] Brandon-Jones, A. and Slack, N. (2008). *Quantitative Analysis in Operations Management*. Financial Times Prentice Hall.

[25] Breton, E. and Bézivin, J. (2001). Towards an understanding of model executability. In *International Conference on Formal Ontology in Information Systems FOIS, Proceedings*, pages 70–80.

[26] Buijs, J. C. A. M., van Dongen, B. F., and van der Aalst, W. M. P. (2012). A genetic algorithm for discovering process trees. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012*, pages 1–8. IEEE.

[27] Burattin, A., Sperduti, A., and van der Aalst, W. M. P. (2012). Heuristics miners for streaming event data. *CoRR*, abs/1212.6383.

[28] Burattin, A., Sperduti, A., and Veluscek, M. (2013). Business models enhancement through discovery of roles. In *IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2013*, pages 103–110. IEEE.

[29] Buzacott, J. A. (1996). Commonalities in reengineered business processes: Models and issues. *Management Science*, 42(5):pp. 768–782.

[30] Caporuscio, M., Marco, A. D., and Inverardi, P. (2005). Run-time performance management of the siena publish/subscribe middleware. In *Proceedings of the Fifth International Workshop on Software and Performance, WOSP 2005*, pages 65–74. ACM.

[31] Caporuscio, M., Marco, A. D., and Inverardi, P. (2007). Model-based system reconfiguration for dynamic performance management. *Journal of Systems and Software*, 80(4):455–473.

[32] Cardoso, J. and Lenic, M. (2006). Web process and workflow path mining using the multimethod approach. *IJBIDM*, 1(3):304–328.

[33] Carmona, J. and Cortadella, J. (2010). Process mining meets abstract interpretation. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Proceedings, Part I*, volume 6321 of *Lecture Notes in Computer Science*, pages 184–199. Springer.

[34] Carmona, J. and Gavaldà, R. (2012). Online techniques for dealing with concept drift in process mining. In *Advances in Intelligent Data Analysis XI - 11th International Symposium, IDA 2012. Proceedings*, volume 7619 of *Lecture Notes in Computer Science*, pages 90–102. Springer.

[35] Carzaniga, A., Gorla, A., Mattavelli, A., Perino, N., and Pezzè, M. (2013). Automatic recovery from runtime failures. In *35th International Conference on Software Engineering, ICSE '13*, pages 782–791. IEEE / ACM.

[36] Casanova, P., Schmerl, B. R., Garlan, D., and Abreu, R. (2011). Architecture-based run-time fault diagnosis. In *Software Architecture - 5th European Conference, ECSA 2011. Proceedings*, volume 6903 of *Lecture Notes in Computer Science*, pages 261–277. Springer.

[37] Cattafi, M., Lamma, E., Riguzzi, F., and Storari, S. (2010). Incremental declarative process mining. In *Smart Information and Knowledge Management: Advances, Challenges, and Critical Issues*, volume 260 of *Studies in Computational Intelligence*, pages 103–127. Springer.

[38] Cheng, B. H. C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Serugendo, G. D. M., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H. M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H. A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., and

Whittle, J. (2009). Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer.

[39] Cook, J. E. and Wolf, A. L. (1998). Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249.

[40] Cortadella, J., Kishinevsky, M., Lavagno, L., and Yakovlev, A. (1998). Deriving petri nets for finite transition systems. *IEEE Trans. Computers*, 47(8):859–882.

[41] Cousot, P. and Cousot, R. (1977). Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages 1977*, pages 238–252. ACM.

[42] Creswell, J. (1998). *Qualitative inquiry and research design: choosing among five traditions*. Sage Publications series. Sage Publications.

[43] Czarnecki, K. (2004). Overview of generative software development. In *Unconventional Programming Paradigms, International Workshop UPP 2004, Revised Selected and Invited Papers*, volume 3566 of *Lecture Notes in Computer Science*, pages 326–341. Springer.

[44] Davenport, T. H. (2010). Business intelligence and organizational decisions. *IJBIR*, 1(1):1–12.

[45] de Medeiros, A. K. A., van Dongen, B. F., van der Aalst, W. M. P., and Weijters, A. J. M. M. (2004). Process mining for ubiquitous mobile systems: An overview and a concrete algorithm. In *Ubiquitous Mobile Information and Collaboration Systems, Second CAiSE Workshop, UMICS 2004, Revised Selected Papers*, volume 3272 of *Lecture Notes in Computer Science*, pages 151–165. Springer.

[46] de Medeiros, A. K. A., Weijters, A. J. M. M., and van der Aalst, W. M. P. (2007). Genetic process mining: an experimental evaluation. *Data Min. Knowl. Discov.*, 14(2):245–304.

[47] Dehnert, J. and van der Aalst, W. M. P. (2004). Bridging the gap between business models and workflow specifications. *Int. J. Cooperative Inf. Syst.*, 13(3):289–332.

[48] Del Fabro, M. D., Bézivin, J., and Valduriez, P. (2006). Weaving models with the eclipse amw plugin. *Eclipse Modeling Symposium, Eclipse Summit Europe*, 2006.

[49] del-Río-Ortega, A., de Reyna, M. R. A., Toro, A. D., and Cortés, A. R. (2012). Defining process performance indicators by using templates and patterns. In *Business Process Management - 10th International Conference, BPM 2012. Proceedings*, volume 7481 of *Lecture Notes in Computer Science*, pages 223–228. Springer.

[50] del-Río-Ortega, A., Resinas, M., and Cortés, A. R. (2010). Defining process performance indicators: An ontological approach. In *On the Move to Meaningful Internet Systems: OTM 2010 - Confederated International Conferences: CoopIS, IS, DOA and ODBASE, Proceedings, Part I*, volume 6426 of *Lecture Notes in Computer Science*, pages 555–572. Springer.

[51] DeRemer, F. and Kron, H. H. (1976). Programming-in-the-large versus programming-in-the-small. *IEEE Trans. Software Eng.*, 2(2):80–86.

[52] Deursen, A. V., Visser, E., and Warmer, J. (2007). Model-driven software evolution: A research agenda. In *IN PROC. INT. WS ON MODEL-DRIVEN SOFTWARE EVOLUTION HELD WITH THE ECSMR'07*. Delft University of Technology.

[53] Dey, A. K. (2001). Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7.

[54] Dodig-Crnkovic, Gordana (2002). Scientific Methods in Computer Science. In *Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden*.

[55] Dresner, H. (2002). Business Activity Monitoring: New Age BI. ID no. G00105565. Gartner Research.

[56] Drobek, M., Gilani, W., Molka, T., and Soban, D. (2015a). Automated equation formulation for causal loop diagrams. In *Business Information Systems - 18th International Conference, BIS 2015, Proceedings*, volume 208 of *Lecture Notes in Business Information Processing*, pages 38–49. Springer.

[57] Drobek, M., Gilani, W., Redlich, D., and Molka, T. (2015b). Model-based business continuity management. US Patent (filed in 2015).

[58] Drobek, M., Gilani, W., Redlich, D., Molka, T., and Soban, D. (2014). Advanced business simulations - incorporating business and process execution data. In *Business Modeling and Software Design - 4th International Symposium, BMSD 2014, Revised Selected Papers*, volume 220 of *Lecture Notes in Business Information Processing*, pages 119–137. Springer.

[59] Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. (2008). *Selecting Empirical Methods for Software Engineering Research*, pages 285–311. Springer London".

[60] Eckert, M. and Bry, F. (2009). Complex event processing (CEP). *Informatik Spektrum*, 32(2):163–167.

[61] Epifani, I., Ghezzi, C., Mirandola, R., and Tamburrelli, G. (2009). Model evolution by run-time parameter adaptation. In *31st International Conference on Software Engineering, ICSE 2009, Proceedings*, pages 111–121. IEEE.

[62] Fleurey, F., Dehlen, V., Bencomo, N., Morin, B., and Jézéquel, J. (2008). Modeling and validating dynamic adaptation. In *Models in Software Engineering, Workshops and Symposia at MODELS 2008. Reports and Revised Selected Papers*, volume 5421 of *Lecture Notes in Computer Science*, pages 97–108. Springer.

[63] Foundation, E. (2015). Eclipse graphical editing framework (gef). http://www.eclipse.org/gef (accessed July 02nd, 2015).

[64] France, R. B. and Rumpe, B. (2003). Editorial - model engineering. *Software and System Modeling*, 2(2):73–75.

[65] France, R. B. and Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. In *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007*, pages 37–54.

[66] Frank, U. (2008). The memo meta modelling language (mml) and language architecture. ICB Research Reports 24, University Duisburg-Essen, Institute for Computer Science and Business Information Systems (ICB).

[67] Frank, U. (2014). Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. *Software and System Modeling*, 13(3):941–962.

[68] Friedenstab, J., Janiesch, C., Matzner, M., and Müller, O. (2012). Extending BPMN for business activity monitoring. In *45th Hawaii International International Conference on Systems Science (HICSS-45 2012), Proceedings, 4-7 January 2012, Grand Wailea, Maui, HI, USA*, pages 4158–4167. IEEE Computer Society.

[69] Fritzsche, M. (2010). *Performance related Decision Support for Process Modelling*. PhD thesis, School of Electronics, Electrical Engineering and Computer Science, Queens University Belfast.

[70] Fritzsche, M., Gilani, W., and Picht, M. (2011). Process-Centric Decision Support. In Rosenberg, A., von Rosing, M., Chase, G., Omar, R., and Taylor, J., editors, *Applying Real-World BPM in an SAP Environment*. SAP PRESS.

[71] Fritzsche, M., Johannes, J., Aßmann, U., Mitschke, S., Gilani, W., Spence, I. T. A., Brown, T. J., and Kilpatrick, P. (2008). Systematic usage of embedded modelling languages in automated model transformation chains. In *Software Language Engineering, First International Conference, SLE 2008. Revised Selected Papers*, volume 5452 of *Lecture Notes in Computer Science*, pages 134–150. Springer.

[72] Fritzsche, M., Johannes, J., Cech, S., and Gilani, W. (2009a). MDPE workbench - A solution for performance related decision support. In *Proceedings of the Business Process Management Demonstration Track (BPMDemos 2009)*, volume 489 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[73] Fritzsche, M., Picht, M., Gilani, W., Spence, I. T. A., Brown, T. J., and Kilpatrick, P. (2009b). Extending BPM environments of your choice with performance related decision support. In *Business Process Management, 7th International Conference, BPM 2009, Proceedings*, volume 5701 of *Lecture Notes in Computer Science*, pages 97–112. Springer.

[74] Galushka, M. and Gilani, W. (2014). Drugfusion - retrieval knowledge management for prediction of adverse drug events. In *Business Information Systems - 17th International Conference, BIS 2014. Proceedings*, volume 176 of *Lecture Notes in Business Information Processing*, pages 13–24. Springer.

[75] Garlan, D., Cheng, S., Huang, A., Schmerl, B. R., and Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37(10):46–54.

[76] Garlan, D. and Schmerl, B. R. (2002). Model-based adaptation for self-healing systems. In *Proceedings of the First Workshop on Self-Healing Systems, WOSS 2002*, pages 27–32. ACM.

[77] Garzon, S. R. and Cebulla, M. (2010). Model-based personalization within an adaptable human-machine interface environment that is capable of learning from user interactions. In *ACHI 2010, The Third International Conference on Advances in Computer-Human Interactions*, pages 191–198. IEEE Computer Society.

[78] Ghezzi, C. (2011). The fading boundary between development time and run time. In *9th IEEE European Conference on Web Services, ECOWS 2011*, page 11. IEEE.

[79] Ghezzi, C., Mocci, A., and Sangiorgio, M. (2011). Runtime monitoring of functional component changes with behavior models. In *Models in Software Engineering - Workshops and Symposia at MODELS 2011, Reports and Revised Selected Papers*, volume 7167 of *Lecture Notes in Computer Science*, pages 152–166. Springer.

[80] Ghezzi, C. and Tamburrelli, G. (2009). Predicting performance properties for open systems with KAMI. In *Architectures for Adaptive Software Systems, 5th International Conference on the Quality of Software Architectures, QoSA 2009, Proceedings*, volume 5581 of *Lecture Notes in Computer Science*, pages 70–85. Springer.

[81] Giese, H., Seibel, A., and Vogel, T. (2009). A Model-Driven Configuration Management System for Advanced IT Service Management. In *Proceedings of the 4th International Workshop on Models@run.time at the 12th IEEE/ACM International Conference on Model Driven Engineering Languages and Systems (MoDELS 2009)*, volume 509 of *CEUR Workshop Proceedings*, pages 61–70. CEUR-WS.org.

[82] Giese, H. and Wagner, R. (2009). From model transformation to incremental bidirectional model synchronization. *Software and System Modeling*, 8(1):21–43.

[83] Gilani, W., Galushka, M., Molka, T., Redlich, D., Du, Y., and Drobek, M. (2014). A process-oriented performance and risk management workbench. In *eChallenges e-2014, 2014 Conference*, pages 1–9.

[84] Gilani, W., Redlich, D., Galushka, M., Molka, T., and Du, Y. (2013). Timbus : Digital preservation for timeless business processes and services. In *eChallenges e-2013, 2013 Conference*.

[85] Gjerlufsen, T., Ingstrup, M., Wolff, J., and Olsen, O. (2009). Mirrors of meaning: Supporting inspectable runtime models. *IEEE Computer*, 42(10):61–68.

[86] Grace, P., Blair, G. S., and Samuel, S. (2005). A reflective framework for discovery and interaction in heterogeneous mobile environments. *Mobile Computing and Communications Review*, 9(1):2–14.

[87] Gregg, D. G., Kulkarni, U. R., and Vinze, A. S. (2001). Understanding the philosophical underpinnings of software engineering research in information systems. *Information Systems Frontiers*, 3(2):169–183.

[88] Günther, C. W., Rinderle, S. B., Reichert, M., van der Aalst, W. M. P., and Recker, J. (2008). Using process mining to learn from process changes in evolutionary systems. *International Journal of Business Process Integration and Management, Special Issue on Business Process Flexibility*, 3(1):61–78.

[89] Günther, C. W. and van der Aalst, W. M. P. (2007). Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In *Business Process Management, 5th International Conference, BPM 2007, Proceedings*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer.

[90] Günther, C. W. and Verbeek, E. (2014). XES Standard Definition Version 2.0. http://www.xes-standard.org/_media/xes/xesstandarddefinition-2.0.pdf (accessed July 13, 2015).

[91] Gupta, N. K., Jagadeesan, L. J., Koutsofios, E., and Weiss, D. M. (1997). Auditdraw: Generating audits the FAST way. In *3rd IEEE International Symposium on Requirements Engineering (RE'97)*, pages 188–197. IEEE Computer Society.

[92] Hailpern, B. and Tarr, P. L. (2006). Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal*, 45(3):451–462.

[93] Hamann, L., Gogolla, M., and Kuhlmann, M. (2011). Ocl-based runtime monitoring of JVM hosted applications. *ECEASST*, 44.

[94] Hamann, L., Hofrichter, O., and Gogolla, M. (2012). Ocl-based runtime monitoring of applications with protocol state machines. In *Modelling Foundations and Applications - 8th European Conference, ECMFA 2012. Proceedings*, volume 7349 of *Lecture Notes in Computer Science*, pages 384–399. Springer.

[95] Harmon, P. (2015). The scope and evolution of business process management. In vom Brocke, J. and Rosemann, M., editors, *Handbook on Business Process Management 1, Introduction, Methods, and Information Systems, 2nd Ed.*, International Handbooks on Information Systems, pages 37–80. Springer.

[96] Herbst, J. and Karagiannis, D. (2000). Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. *Int. Syst. in Accounting, Finance and Management*, 9(2):67–92.

[97] Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 28(1):75–105.

[98] Hill, J. B., Pezzini, M., and Natis, Y. V. (2008). Findings: Confusion remains regarding BPM terminologies. ID no. G00155817. Gartner Research.

[99] Hooman, J. and Hendriks, T. (2007). Model-based run-time error detection. In *Models in Software Engineering, Workshops and Symposia at MoDELS 2007, Reports and Revised Selected Papers*, volume 5002 of *Lecture Notes in Computer Science*, pages 225–236. Springer.

[100] Huang, G., Song, H., and Mei, H. (2009). Sm@rt: Towards architecture-based runtime management of internetware systems. In *Proceedings of the First Asia-Pacific Symposium on Internetware*, Internetware '09, pages 9:1–9:10, New York, NY, USA. ACM.

[101] Hummer, W., Gaubatz, P., Strembeck, M., Zdun, U., and Dustdar, S. (2011). An integrated approach for identity and access management in a SOA context. In *SACMAT 2011, 16th ACM Symposium on Access Control Models and Technologies, Proceedings*, pages 21–30. ACM.

[102] IBM Redbooks (2002). *IBM WebSphere V4.0 Advanced Edition Handbook*. IBM.

[103] Intalio (2014). Intalio BPMS. http://www.intalio.com/products/bpms/overview/ (accessed Dec. 6th, 2014).

[104] International SEMATECH and National Institute of Standards and Technology (2013). Engineering statistics handbook. http://www.itl.nist.gov/div898/handbook/.

[105] Inverardi, P. and Mori, M. (2013). A software lifecycle process to support consistent evolutions. In *Software Engineering for Self-Adaptive Systems II - International Seminar 2010, Revised Selected and Invited Papers*, volume 7475 of *Lecture Notes in Computer Science*, pages 239–264. Springer.

[106] Janiesch, C., Matzner, M., and Müller, O. (2012). Beyond process monitoring: a proof-of-concept of event-driven business activity management. *Business Proc. Manag. Journal*, 18(4):625–643.

[107] Janiesch, C., Matzner, M., Müller, O., Vollmer, R., and Becker, J. (2011). Slipstream: architecture options for real-time process analytics. In *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC)*, pages 295–300. ACM.

[108] Jaring, M. (2005). *Variability Engineering as an Integral Part of the Software Product Family Development Process*. PhD thesis, Institute of Mathematics and Computing Science, University of Groningen.

[109] Jensen, K. (1997). *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use - Volume 3*. Monographs in Theoretical Computer Science. An EATCS Series. Springer.

[110] Johanndeiter, T., Goldstein, A., and Frank, U. (2013). Towards business process models at runtime. In *Proceedings of the 8th Workshop on Models @ Run.time co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013)*, volume 1079 of *CEUR Workshop Proceedings*, pages 13–25. CEUR-WS.org.

[111] Jouault, F. (2005). Loosely coupled traceability for atl. In *In Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability*, pages 29–37.

[112] Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). ATL: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39.

[113] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., and Valduriez, P. (2006a). ATL: a qvt-like transformation language. In *Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006*, pages 719–720. ACM.

[114] Jouault, F., Bézivin, J., and Kurtev, I. (2006b). TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In *Generative Programming and Component Engineering, 5th International Conference, GPCE 2006, Proceedings*, pages 249–254. ACM.

[115] Kent, S. (2002). Model driven engineering. In *Integrated Formal Methods, Third International Conference, IFM 2002, Proceedings*, volume 2335 of *Lecture Notes in Computer Science*, pages 286–298. Springer.

[116] Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *IEEE Computer*, 36(1):41–50.

[117] Kieburtz, R. B., McKinney, L., Bell, J. M., Hook, J., Kotov, A., Lewis, J., Oliva, D., Sheard, T., Smith, I., and Walton, L. (1996). A software engineering experiment in software component generation. In *18th International Conference on Software Engineering, Proceedings*, pages 542–552. IEEE Computer Society.

[118] Kiepuszewski, B., ter Hofstede, A. H. M., and Bussler, C. (2000). On structured workflow modelling. In *Advanced Information Systems Engineering, 12th International Conference CAiSE 2000, Proceedings*, volume 1789 of *Lecture Notes in Computer Science*, pages 431–445. Springer.

[119] Kindler, E., Rubin, V., and Schäfer, W. (2005). Incremental workflow mining based on document versioning information. In *Unifying the Software Process Spectrum, International Software Process Workshop, SPW 2005, Revised Selected Papers*, volume 3840 of *Lecture Notes in Computer Science*, pages 287–301. Springer.

[120] Kindler, E., Rubin, V., and Schäfer, W. (2006a). Activity mining for discovering software process models. In *Software Engineering 2006, Fachtagung des GI-Fachbereichs Softwaretechnik*, volume 79 of *LNI*, pages 175–180. GI.

[121] Kindler, E., Rubin, V., and Schäfer, W. (2006b). Incremental workflow mining for process flexibility. In *Proceedings of the CAISE*06 Workshop on Business Process Modelling, Development, and Support BPMDS 2006*, volume 236 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[122] Ko, R. K. L. (2009). A computer scientist's introductory guide to business process management (BPM). *ACM Crossroads*, 15(4):4:11–4:18.

[123] Ko, R. K. L., Lee, S. S. G., and Lee, E. W. (2009). Business Process Management (BPM) Standards: A Survey. *Business Proc. Manag. Journal*, 15(5):744–791.

[124] Krueger, C. W. (1992). Software reuse. *ACM Comput. Surv.*, 24(2):131–183.

[125] Lawrence, P. (1997). *Workflow Handbook 1997, Workflow Management Coalition*. John Wiley and Sons, New York.

[126] Leemans, S. J. J., Fahland, D., and van der Aalst, W. M. P. (2013a). Discovering block-structured process models from event logs - A constructive approach. In *Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings*, volume 7927 of *Lecture Notes in Computer Science*, pages 311–329. Springer.

[127] Leemans, S. J. J., Fahland, D., and van der Aalst, W. M. P. (2013b). Discovering block-structured process models from event logs containing infrequent behaviour. In *Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, volume 171 of *Lecture Notes in Business Information Processing*, pages 66–78. Springer.

[128] Leemans, S. J. J., Fahland, D., and van der Aalst, W. M. P. (2014). Discovering block-structured process models from incomplete event logs. In *Application and Theory of Petri Nets and Concurrency - 35th International Conference, PETRI NETS 2014. Proceedings*, volume 8489 of *Lecture Notes in Computer Science*, pages 91–110. Springer.

[129] Lehmann, G., Blumendorf, M., Trollmann, F., and Albayrak, S. (2010). Meta-modeling runtime models. In *Models in Software Engineering - Workshops and Symposia at MODELS 2010, Reports and Revised Selected Papers*, volume 6627 of *Lecture Notes in Computer Science*, pages 209–223. Springer.

[130] Liu, R., Nigam, A., Jeng, J., Shieh, C., and Wu, F. Y. (2010). Integrated modeling of performance monitoring with business artifacts. In *IEEE 7th International Conference on e-Business Engineering, ICEBE 2010*, pages 64–71. IEEE Computer Society.

[131] Liu, Y., Zhang, H., Li, C., and Jiao, R. J. (2012). Workflow simulation for operational decision support using event graph through process mining. *Decision Support Systems*, 52(3):685–697.

[132] Lorenz, R., Mauser, S., and Juhás, G. (2007). How to synthesize nets from languages: a survey. In *Proceedings of the Winter Simulation Conference, WSC 2007*, pages 637–647. WSC.

[133] Lu, R. and Sadiq, S. W. (2007). A survey of comparative business process modeling approaches. In *Business Information Systems, 10th International Conference, BIS 2007, Proceedings*, volume 4439 of *Lecture Notes in Computer Science*, pages 82–94. Springer.

[134] Luckham, D. C. (2005). *The power of events - an introduction to complex event processing in distributed enterprise systems*. ACM.

[135] Ludewig, J. (2004). Models in software engineering - an introduction. *Inform., Forsch. Entwickl.*, 18(3-4):105–112.

[136] Ly, L. T., Rinderle, S., Dadam, P., and Reichert, M. (2005). Mining staff assignment rules from event-based data. In *Business Process Management Workshops, BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS, Revised Selected Papers*, volume 3812, pages 177–190.

[137] Maes, P. (1987). Concepts and experiments in computational reflection. In *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'87), 1987, Proceedings*, pages 147–155. ACM.

[138] Maggi, F. M., Burattin, A., Cimitile, M., and Sperduti, A. (2013). Online process discovery to detect concept drifts in ltl-based declarative process models. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences - Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE 2013. Proceedings*, volume 8185 of *Lecture Notes in Computer Science*, pages 94–111. Springer.

[139] Manku, G. S. and Motwani, R. (2002). Approximate frequency counts over data streams. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases*, pages 346–357. Morgan Kaufmann.

[140] Mannila, H., Toivonen, H., and Verkamo, A. I. (1995). Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95), 1995*, pages 210–215. AAAI Press.

[141] Maoz, S. (2009). Using model-based traces as runtime models. *IEEE Computer*, 42(10):28–36.

[142] Maoz, S. and Harel, D. (2011). On tracing reactive systems. *Software and System Modeling*, 10(4):447–468.

[143] March, S. T., Hevner, A. R., and Ram, S. (2000). Research commentary: An agenda for information technology research in heterogeneous and distributed environments. *Information Systems Research*, 11(4):327–341.

[144] Mendling, J., Neumann, G., and Nüttgens, M. (2004). A Comparison of XML Interchange Formats for Business Process Modelling. In *EMISA 2004, Informationssysteme im E-Business und E-Government*, volume 56 of *LNI*, pages 129–140. GI.

[145] Milner, R. (1999). *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press.

[146] Molka, T., Redlich, D., Drobek, M., Caetano, A., Zeng, X., and Gilani, W. (2014). Conformance checking for bpmn-based process models. In *Symposium on Applied Computing, SAC 2014*, pages 1406–1413. ACM.

[147] Molka, T., Redlich, D., Drobek, M., Zeng, X., and Gilani, W. (2015a). Diversity guided evolutionary mining of hierarchical process models. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015*, pages 1247–1254. ACM.

[148] Molka, T., Redlich, D., Gilani, W., Zeng, X., and Drobek, M. (2015b). Evolutionary computation based discovery of hierarchical business process models. In *Business Information Systems - 18th International Conference, BIS 2015, Proceedings*, volume 208 of *Lecture Notes in Business Information Processing*, pages 191–204. Springer.

[149] Momm, C., Malec, R., and Abeck, S. (2007). Towards a model-driven development of monitored processes. In *eOrganisation: Service-, Prozess-, Market-Engineering: 8. Internationale Tagung Wirtschaftsinformatik - Band 2, WI 2007*, pages 319–336. Universitaetsverlag Karlsruhe.

[150] Monteiro, L. F. S. and de Oliveira, K. M. (2011). Defining a catalog of indicators to support process performance analysis. *Journal of Software Maintenance*, 23(6):395–422.

[151] Morin, B., Barais, O., Jézéquel, J., Fleurey, F., and Solberg, A. (2009). Models@run.time to support dynamic adaptation. *IEEE Computer*, 42(10):44–51.

[152] Mos, A. and Murphy, J. (2002). Performance management in component-oriented systems using a model driven architecture^{tm} approach. In *6th International Enterprise Distributed Object Computing Conference (EDOC 2002), Proceedings*, pages 227–237. IEEE Computer Society.

[153] Muller, P., Fleurey, F., and Jézéquel, J. (2005). Weaving executability into object-oriented meta-languages. In *Model Driven Engineering Languages and Systems, 8th International Conference, MoDELS 2005, Proceedings*, volume 3713 of *Lecture Notes in Computer Science*, pages 264–278. Springer.

[154] Müller, R., Greiner, U., and Rahm, E. (2004). A$_{gent}$W$_{ork}$: a workflow system supporting rule-based workflow adaptation. *Data Knowl. Eng.*, 51(2):223–256.

[155] Munoz-Gama, J. and Carmona, J. (2010). A fresh look at precision in process conformance. In *Business Process Management - 8th International Conference, BPM 2010. Proceedings*, volume 6336 of *Lecture Notes in Computer Science*, pages 211–226. Springer.

[156] Muskens, J. and Chaudron, M. R. V. (2004). Integrity management in component based systems. In *30th EUROMICRO Conference 2004*, pages 611–619. IEEE Computer Society.

[157] Nguyen, V. H., Fouquet, F., Plouzeau, N., and Barais, O. (2012). A process for continuous validation of self-adapting component based systems. In *Proceedings of the 7th Workshop on Models@run.time, 2012*, pages 32–37. ACM.

[158] Nierstrasz, O., Denker, M., and Renggli, L. (2009). Model-centric, context-aware software adaptation. In *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*, volume 5525 of *Lecture Notes in Computer Science*, pages 128–145. Springer.

[159] Nordstrom, G., Sztipanovits, J., Karsai, G., and Lédeczi, Á. (1999). Metamodeling - rapid design and evolution of domain-specific modeling environments. In *6th Symposium on Engineering of Computer-Based Systems (ECBS '99), IEEE Computer Society, 1999*, pages 68–74. IEEE Computer Society.

[160] Nordstrom, S., Dubey, A., Keskinpala, T., Datta, R., Neema, S., and Bapty, T. (2007). Model predictive analysis for autonomicworkflow management in large-scale scientific computing environments. In *Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems, 2007. EASe'07*, pages 37–42. IEEE.

[161] OASIS (2007). Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[162] Object Management Group (2006). Meta Object Facility (MOF) specification version 2.0. http://www.omg.org/cgi-bin/doc?formal/2006-01-01 (accessed July 13, 2015).

[163] Object Management Group (2015). Meta Object Facility (MOF) 2.0 - Query/View/Transformation Specification. http://www.omg.org/spec/QVT/1.2/ (accessed June 14th 2015).

[164] Object Management Group Inc (2003). BPMN and Business Process Management - Introduction to the New Business Process Modeling Standard. http://www.omg.org/bpmn/Documents/6AD5D16960.BPMN_and_BPM.pdf (accessed Dec. 12, 2014).

[165] Object Management Group Inc (2005). Unified Modeling Language 2.0: Superstructure. http://www.omg.org/spec/UML/2.0/Superstructure/PDF. formal/05-07-04.

[166] Object Management Group Inc (2007). The OMG Business Process Related Standards. http://bpmfocus.pbworks.com/f/BPM+Standards+At+The+OMG+-+July+07.pdf (accessed Dec. 12, 2014).

[167] Object Management Group Inc (2008). Business Process Definition Meta-Model - Volume I: Common Infrastructure. http://www.omg.org/spec/BPDM/1.0. formal/2008-11-03.

[168] Object Management Group Inc (2011). Business Process Model and Notation (BPMN) Specification 2.0. http://www.omg.org/spec/BPMN/2.0/PDF. formal/2011-01-03.

[169] Object Management Group Inc (2015). Model-driven architecture. http://www.omg.org/mda/specs.htm (accessed July 02nd, 2015).

[170] Otter, M., Mattsson, S. E., and Elmqvist, H. (2007). Multidomain modeling with modelica. In *Handbook of Dynamic System Modeling*. Chapman and Hall/CRC.

[171] Pesic, M., Schonenberg, M. H., Sidorova, N., and van der Aalst, W. M. P. (2007). Constraint-Based Workflow Models: Change Made Easy. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences, Proceedings, Part I*, volume 4803 of *Lecture Notes in Computer Science*, pages 77–94. Springer.

[172] Petri, C. A. (1962). *Kommunikation mit Automaten*. PhD thesis, Rheinisch-Westfälisches Institut f. instrumentelle Mathematik.

[173] Pidd, M. (1998). *Computer Simulation in Management Science*. John Wiley & Sons, Inc., New York, NY, USA, 4th edition.

[174] Pienaar, J. A., Raghunathan, A., and Chakradhar, S. T. (2011). MDR: performance model driven runtime for heterogeneous parallel platforms. In *Proceedings of the 25th International Conference on Supercomputing, 2011*, pages 225–234. ACM.

[175] Polyvyanyy, A., García-Bañuelos, L., Fahland, D., and Weske, M. (2014). Maximal structuring of acyclic process models. *Comput. J.*, 57(1):12–35.

[176] Poole, J. D. (2001). Model-driven architecture: Vision, standards and emerging technologies. In *ECOOP 2001 - Workshop on Metamodeling and Adaptive Object Models*.

[177] Popper, K. (2002). *Conjectures and Refutations: The Growth of Scientific Knowledge*. Classics Series. Routledge.

[178] Porzucek, T., Kluth, S., Fritzsche, M., and Redlich, D. (2010). Combination of a discrete event simulation and an analytical performance analysis through model-transformations. In *17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS 2010*, pages 183–192. IEEE Computer Society.

[179] Raffelsieper, T., Becker, J., Matzner, M., and Janiesch, C. (2012). Requirements for a pattern language for event-driven business activity monitoring. In *Multikonferenz Wirtschaftsinformatik 2012*, pages 77–89. BoD.

[180] Rajsiri, V., Fleury, N., Crosmarie, G., and Lorré, J. (2010). Event-based business process editor and simulator. In *Business Process Management Workshops - BPM 2010 International Workshops and Education Track, Revised Selected Papers*, volume 66 of *Lecture Notes in Business Information Processing*, pages 707–718. Springer.

[181] Redlich, D., Blair, G. S., Rashid, A., Molka, T., and Gilani, W. (2011). Research challenges for business process models at run-time. In *Models@run.time - Foundations, Applications, and Roadmaps [Dagstuhl Seminar 11481, 2011].*, volume 8378 of *Lecture Notes in Computer Science*, pages 208–236. Springer.

[182] Redlich, D., Galushka, M., Molka, T., Gilani, W., Blair, G. S., and Rashid, A. (2015a). Evaluation of the dynamic construct competition miner for an ehealth system. In *Business Information Systems - 18th International Conference, BIS 2015, Proceedings*, volume 208 of *Lecture Notes in Business Information Processing*, pages 115–126. Springer.

[183] Redlich, D. and Gilani, W. (2011). Event-driven process-centric performance prediction via simulation. In *Business Process Management Workshops - BPM 2011 International Workshops, Revised Selected Papers, Part I*, volume 99 of *Lecture Notes in Business Information Processing*, pages 473–478. Springer.

[184] Redlich, D., Gilani, W., Molka, T., Drobek, M., Rashid, A., and Blair, G. S. (2014a). Introducing a framework for scalable dynamic process discovery. In *Advances in Enterprise Engineering VIII - 4th Enterprise Engineering Working Conference, EEWC 2014. Proceedings*, volume 174 of *Lecture Notes in Business Information Processing*, pages 151–166. Springer.

[185] Redlich, D., Molka, T., Gilani, W., Blair, G. S., and Rashid, A. (2014b). Constructs competition miner: Process control-flow discovery of bp-domain constructs. In *Business Process Management - 12th International Conference, BPM 2014. Proceedings*, volume 8659 of *Lecture Notes in Computer Science*, pages 134–150. Springer.

[186] Redlich, D., Molka, T., Gilani, W., Blair, G. S., and Rashid, A. (2014c). Scalable dynamic business process discovery with the constructs competition miner. In *Proceedings of the 4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2014), 2014.*, volume 1293 of *CEUR Workshop Proceedings*, pages 91–107. CEUR-WS.org.

[187] Redlich, D., Molka, T., Gilani, W., Blair, G. S., and Rashid, A. (2015b). Dynamic constructs competition miner - occurrence- vs. time-based ageing. In *4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2014), Revised Selected Papers*, volume 237 of *Lecture Notes in Business Information Processing*, pages 79–106. Springer.

[188] Redlich, D., Platz, S., and Gilani, W. (2012). Mde in practice: Process-centric performance prediction via simulation in real-time. In *Joint Proceedings of co-located Events at the 8th European Conference on Modelling Foundations and Applications(ECMFA) - Tool Presentations*, pages 336–339.

[189] Redlich, D., Winkler, U., Molka, T., and Gilani, W. (2014d). Model-driven engineering in practice: integrated performance decision support for process-centric business impact analysis. In *ACM/SPEC International Conference on Performance Engineering, ICPE 2014*, pages 247–258. ACM.

[190] Regev, G., Soffer, P., and Schmidt, R. (2006). Taxonomy of Flexibility in Business Processes. In *Proceedings of the CAISE*06 Workshop on Business Process Modelling, Development, and Support BPMDS*, volume 236 of *CEUR Workshop Proceedings*.

[191] Reichert, M. and Dadam, P. (1998). Adept Flex: Supporting Dynamic Changes of Workflows Without Losing Control. *J. Intell. Inf. Syst.*, 10(2):93–129.

[192] Reichert, M. and Weber, B. (2012). *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer.

[193] Robinson, S. (2004). *Simulation: The Practice of Model Development and Use*. John Wiley & Sons.

[194] Rosemann, M. and vom Brocke, J. (2015). The six core elements of business process management. In vom Brocke, J. and Rosemann, M., editors, *Handbook on Business Process Management 1, Introduction, Methods, and Information Systems, 2nd Ed.*, International Handbooks on Information Systems, pages 105–122. Springer.

[195] Roser, C., Nakano, M., and Tanaka, M. (2001). A practical bottleneck detection method. In *Proceedings of the 33nd conference on Winter simulation, WSC 2001*, pages 949–953. ACM.

[196] Rothenberg, J. (1989). Artificial intelligence, simulation & modeling. chapter The Nature of Modeling, pages 75–92. John Wiley & Sons, Inc., New York, NY, USA.

[197] Rozinat, A., Mans, R. S., Song, M., and van der Aalst, W. M. P. (2008a). Discovering colored petri nets from event logs. *STTT*, 10(1):57–74.

[198] Rozinat, A., Mans, R. S., Song, M., and van der Aalst, W. M. P. (2009a). Discovering simulation models. *Inf. Syst.*, 34(3):305–327.

[199] Rozinat, A. and van der Aalst, W. M. P. (2006). Decision mining in prom. In *Business Process Management, 4th International Conference, BPM 2006, Proceedings*, volume 4102 of *Lecture Notes in Computer Science*, pages 420–425. Springer.

[200] Rozinat, A., Wynn, M. T., van der Aalst, W. M. P., ter Hofstede, A. H. M., and Fidge, C. J. (2008b). Workflow simulation for operational decision support using design, historic and state information. In *Business Process Management, 6th International Conference, BPM 2008. Proceedings*, volume 5240 of *Lecture Notes in Computer Science*, pages 196–211. Springer.

[201] Rozinat, A., Wynn, M. T., van der Aalst, W. M. P., ter Hofstede, A. H. M., and Fidge, C. J. (2009b). Workflow simulation for operational decision support. *Data Knowl. Eng.*, 68(9):834–850.

[202] Sadiq, S., Sadiq, W., and Orlowska, M. (2001). Pockets of Flexibility in Workflow Specifications. In *20th International Conference on Conceptual Modeling, ER2001*, pages 513–526.

[203] Sadiq, S. W. (2000). Handling dynamic schema change in process models. In *Proceedings of the Australasian Database Conference*, ADC '00, pages 120–126, Washington, DC, USA. IEEE Computer Society.

[204] Sadiq, S. W., Orlowska, M. E., and Sadiq, W. (2005). Specification and validation of process constraints for flexible workflows. *Inf. Syst.*, 30(5):349–378.

[205] SAP (2014a). BusinessObjects BI Platform. http://www.sap.com/pc/analytics/business-intelligence/software/bi-platform-analytics/ (accessed June 16, 2015).

[206] SAP (2014b). Operational Process Intelligence 1.0. http://help.sap.com/hana-opint (accessed June 16, 2015).

[207] Scheer, I.D.S. (1992). ARIS (Architecture of integrated Information Systems).

[208] Schmerl, B. R., Aldrich, J., Garlan, D., Kazman, R., and Yan, H. (2006). Discovering architectures from running systems. *IEEE Trans. Software Eng.*, 32(7):454–466.

[209] Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39(2):25–31.

[210] Schonenberg, H., Jian, J., Sidorova, N., and van der Aalst, W. M. P. (2010). Business trend analysis by simulation. In *Advanced Information Systems Engineering, 22nd International Conference, CAiSE 2010. Proceedings*, volume 6051 of *Lecture Notes in Computer Science*, pages 515–529. Springer.

[211] Schonenberg, H., Mans, R., Russell, N., Mulyar, N., and van der Aalst, W. M. P. (2008). Process flexibility: A survey of contemporary approaches. In *Advances in Enterprise Engineering I, 4th International Workshop CIAO! and 4th International Workshop EOMAS*, volume 10 of *Lecture Notes in Business Information Processing*, pages 16–30. Springer.

[212] Seidewitz, E. (2003). What models mean. *IEEE Software*, 20(5):26–32.

[213] Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25.

[214] Simchi-Levi, D., Simchi-Levi, E., and Kaminsky, P. (1999). *Designing and Managing the Supply Chain: Concepts, Strategies, and Cases*. McGraw-Hill United-States, New York.

[215] Simmonds, J., Ben-David, S., and Chechik, M. (2010). Monitoring and recovery of web service applications. In *The Smart Internet - Current Research and Future Applications*, volume 6400 of *Lecture Notes in Computer Science*, pages 250–288. Springer.

[216] Simon, H. A. (1996). *The Sciences of the Artificial (3rd Ed.)*. MIT Press, Cambridge, MA, USA.

[217] Sobral, J. L. and Monteiro, M. P. (2008). A domain-specific language for parallel and grid computing. In *Proceedings of the 2008 AOSD workshop on Domain-specific aspect languages*. ACM.

[218] Software AG and Charles Sturt University (2014). Aris standards and conventions manual. https://www.csu.edu.au/__data/assets/pdf_file/0020/1314173/CSU_ARIS_Modelling_Standards_and_Conventions_Manual_V1_0.pdf (accessed November 02th 2017).

[219] Solé, M. and Carmona, J. (2010). Incremental process mining. In *Proceedings of the Workshops of the 31st International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (PETRI NETS 2010) and of the 10th International Conference on Application of Concurrency to System Design (ACSD 2010)*, volume 827 of *CEUR Workshop Proceedings*, pages 175–190. CEUR-WS.org.

[220] Solomon, A. and Litoiu, M. (2011). Business process performance prediction on a tracked simulation model. In *Proceedings of the 3rd International Workshop on Principles of Engineering Service-Oriented Systems*, PESOS '11, pages 50–56, New York, NY, USA. ACM.

[221] Solomon, A., Litoiu, M., Benayon, J., and Lau, A. (2010). Business process adaptation on a tracked simulation model. In *Proceedings of the 2010 conference of the Centre for Advanced Studies on Collaborative Research*, pages 184–198. ACM.

[222] Song, H., Huang, G., Chauvel, F., Xiong, Y., Hu, Z., Sun, Y., and Mei, H. (2011a). Supporting runtime software architecture: A bidirectional-transformation-based approach. *Journal of Systems and Software*, 84(5):711–723.

[223] Song, H., Huang, G., Chauvel, F., Zhang, W., Sun, Y., Shao, W., and Mei, H. (2011b). Instant and incremental QVT transformation for runtime models. In *Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011. Proceedings*, volume 6981 of *Lecture Notes in Computer Science*, pages 273–288. Springer.

[224] Stachowiak, H. (1973). *Allgemeine Modelltheorie*. Springer-Verlag.

[225] Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition.

[226] Striewe, M., Balz, M., and Goedicke, M. (2008). Embedding state machine models in object-oriented source code. *Proceedings of the 3rd Workshop on Models@run.time at MODELS 2008*, pages 6–15.

[227] Szvetits, M. and Zdun, U. (2013). Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. *Software & Systems Modeling*, pages 1–39.

[228] Tsymbal, A. (2004). The problem of concept drift: definitions and related work.

[229] Turner, C. J., Tiwari, A., and Mehnen, J. (2008). A genetic programming approach to business process mining. In *Genetic and Evolutionary Computation Conference, GECCO 2008, Proceedings*, pages 1307–1314. ACM.

[230] van der Aalst, W. M. P. (2007a). Challenges in business process analysis. In *Enterprise Information Systems, 9th International Conference, ICEIS 2007, Revised Selected Papers*, volume 12 of *Lecture Notes in Business Information Processing*, pages 27–42. Springer.

[231] van der Aalst, W. M. P. (2007b). Trends in Business Process Analysis - from Verification to Process Mining. In *ICEIS 2007 - Proceedings of the Ninth International Conference on Enterprise Information Systems*, pages 5–9.

[232] van der Aalst, W. M. P. (2010). Business process simulation revisited. In *Enterprise and Organizational Modeling and Simulation - 6th International Workshop, EOMAS 2010, held at CAiSE 2010. Selected Papers*, volume 63 of *Lecture Notes in Business Information Processing*, pages 1–14. Springer.

[233] van der Aalst, W. M. P. (2011a). *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer.

[234] van der Aalst, W. M. P. (2011b). Process mining: discovering and improving spaghetti and lasagna processes. In *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*, pages 13–20.

[235] van der Aalst, W. M. P. (2015). Business process simulation survival guide. In *Handbook on Business Process Management 1, Introduction, Methods, and Information Systems, 2nd Ed.*, International Handbooks on Information Systems, pages 337–370. Springer.

[236] van der Aalst, W. M. P., Adriansyah, A., and van Dongen, B. F. (2012). Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 2(2):182–192.

[237] van der Aalst, W. M. P., Barros, A. P., ter Hofstede, A. H. M., and Kiepuszewski, B. (2000). Advanced Workflow Patterns. In *Cooperative Information Systems, 7th International Conference, CoopIS 2000*, volume 1901 of *Lecture Notes in Computer Science*, pages 18–29. Springer.

[238] van der Aalst, W. M. P. and Jablonski, S. (2000). Dealing with workflow change: identification of issues and solutions. *International Journal of Computer Systems Science and Engineering*, 15(5):267–276.

[239] van der Aalst, W. M. P., Nakatumba, J., Rozinat, A., and Russell, N. (2008). Business process simulation : how to get it right? Technical Report BPMcenter.org, Brisbane/Eindhoven.

[240] van der Aalst, W. M. P., Pesic, M., and Schonenberg, H. (2009). Declarative Workflows: Balancing between Flexibility and Support. *Computer Science - R&D*, 23(2):99–113.

[241] van der Aalst, W. M. P., Pesic, M., and Song, M. (2010a). Beyond process mining: From the past to present and future. In *Advanced Information Systems Engineering, 22nd International Conference, CAiSE 2010. Proceedings*, volume 6051 of *Lecture Notes in Computer Science*, pages 38–52. Springer.

[242] van der Aalst, W. M. P., Reijers, H. A., and Song, M. (2005a). Discovering social networks from event logs. *Computer Supported Cooperative Work*, 14(6):549–593.

[243] van der Aalst, W. M. P., Rubin, V., Verbeek, H. M. W., van Dongen, B. F., Kindler, E., and Günther, C. W. (2010b). Process mining: a two-step approach to balance between underfitting and overfitting. *Software and System Modeling*, 9(1):87–111.

[244] van der Aalst, W. M. P., Schonenberg, M. H., and Song, M. (2011). Time prediction based on process mining. *Inf. Syst.*, 36(2):450–475.

[245] van der Aalst, W. M. P. and ter Hofstede, A. H. M. (2005). YAWL: Yet Another Workflow Language. *Inf. Syst.*, 30(4):245–275.

[246] van der Aalst, W. M. P., ter Hofstede, A. H. M., and Weske, M. (2003). Business Process Management: A Survey. In *Business Process Management, International Conference, BPM 2003, Proceedings*, volume 2678 of *Lecture Notes in Computer Science*, pages 1–12. Springer.

[247] van der Aalst, W. M. P. and van Hee, K. M. (1995). Framework for business process redesign. In *4th Workshop on Enabling Technologies, Infrastructure for Collaborative Enterprises (WET-ICE'96), Proceedings*, pages 36–45. IEEE Computer Society.

[248] van der Aalst, W. M. P. and Weijters, A. J. M. M. (2004). Process mining: a research agenda. *Computers in Industry*, 53(3):231–244.

[249] van der Aalst, W. M. P., Weijters, T., and Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142.

[250] van der Aalst, W. M. P., Weske, M., and Grünbauer, D. (2005b). Case handling: a new paradigm for business process support. *Data Knowl. Eng.*, 53(2):129–162.

[251] van der Aalst, Wil et al. (2012). Process Mining Manifesto. In Daniel, F., Barkaoui, K., and Dustdar, S., editors, *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer.

[252] van Dongen, B. F., Crooy, R. A., and van der Aalst, W. M. P. (2008). Cycle time prediction: When will this case finally be finished? In *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Proceedings, Part I*, volume 5331 of *Lecture Notes in Computer Science*, pages 319–336. Springer.

[253] van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H. M. W., Weijters, A. J. M. M., and van der Aalst, W. M. P. (2005). The prom framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Proceedings*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer.

[254] van Dongen, B. F., de Medeiros, A. K. A., and Wen, L. (2009). Process mining: Overview and outlook of petri net discovery algorithms. *T. Petri Nets and Other Models of Concurrency II*, 5460:225–242.

[255] van Dongen, B. F. and van der Aalst, W. M. P. (2004). Multi-phase process mining: Building instance graphs. In *Conceptual Modeling - ER 2004, 23rd International Conference on Conceptual Modeling, Proceedings*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer.

[256] van Dongen, B. F. and van der Aalst, W. M. P. (2005). A meta model for process mining data. In *Proceedings of the CAiSE'05 Workshops (EMOI-INTEROP Workshop)*, volume 160 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[257] Vázquez-Barreiros, B., Mucientes, M., and Lama, M. (2014). A genetic algorithm for process discovery guided by completeness, precision and simplicity. In *Business Process Management - 12th International Conference, BPM 2014. Proceedings*, volume 8659 of *Lecture Notes in Computer Science*, pages 118–133. Springer.

[258] Verbeek, H. M. W., Buijs, J. C. A. M., van Dongen, B. F., and van der Aalst, W. M. P. (2010). Xes, xesame, and prom 6. In *Information Systems Evolution - CAiSE Forum 2010, Selected Extended Papers*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75. Springer.

[259] Verbeek, H. M. W. E., Basten, T., and van der Aalst, W. M. P. (2001). Diagnosing workflow processes using woflan. *Comput. J.*, 44(4):246–279.

[260] Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., and Carrasco, R. C. (2005a). Probabilistic finite-state machines-part I. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1013–1025.

[261] Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., and Carrasco, R. C. (2005b). Probabilistic finite-state machines-part II. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1026–1039.

[262] Vogel, T. and Giese, H. (2010). Adaptation and abstract runtime models. In *2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2010*, pages 39–48. ACM.

[263] Vogel, T. and Giese, H. (2014a). Model-driven engineering of self-adaptive software with EUREMA. *TAAS*, 8(4):18.

[264] Vogel, T. and Giese, H. (2014b). On unifying development models and runtime models. In *Proceedings of the 9th Workshop on Models@run.time co-located with 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014)*, volume 1270 of *CEUR Workshop Proceedings*, pages 5–10. CEUR-WS.org.

[265] Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., and Becker, B. (2009a). Incremental model synchronization for efficient run-time monitoring. In *Models in Software Engineering, Workshops and Symposia at MODELS 2009, Reports and Revised Selected Papers*, volume 6002 of *Lecture Notes in Computer Science*, pages 124–139. Springer.

[266] Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., and Becker, B. (2009b). Model-driven architectural monitoring and adaptation for autonomic systems. In *Proceedings of the 6th International Conference on Autonomic Computing, ICAC 2009*, pages 67–68. ACM.

[267] Vogel, T., Seibel, A., and Giese, H. (2010). The role of models and megamodels at runtime. In *Models in Software Engineering - Workshops and Symposia at MODELS 2010, Reports and Revised Selected Papers*, volume 6627 of *Lecture Notes in Computer Science*, pages 224–238. Springer.

[268] von Ammon, R. (2009). Event-driven business process management. In *Encyclopedia of Database Systems*, pages 1068–1071. Springer US.

[269] von Ammon, R., Emmersberger, C., Ertlmaier, T., Etzion, O., Paulus, T., and Springer, F. (2009a). Existing and future standards for event-driven business process management. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS 2009*. ACM.

[270] von Ammon, R., Ertlmaier, T., Etzion, O., Kofman, A., and Paulus, T. (2009b). Integrating Complex Events for Collaborating and Dynamically Changing Business Processes. In *ICSOC/ServiceWave 2009, Revised Selected Papers*, volume 6275 of *Lecture Notes in Computer Science*, pages 370–384.

[271] Waignier, G., Meur, A. L., and Duchien, L. (2009). A model-based framework to design and debug safe component-based autonomic systems. In *Architectures for Adaptive Software Systems, 5th International Conference on the Quality of Software Architectures, QoSA 2009, Proceedings*, volume 5581 of *Lecture Notes in Computer Science*, pages 1–17. Springer.

[272] Weber, B., Reichert, M., and Rinderle-Ma, S. (2008). Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data Knowl. Eng.*, 66(3):438–466.

[273] Weber, B., Rinderle, S., and Reichert, M. (2007). Change patterns and change support features in process-aware information systems. In *Advanced Information Systems Engineering, 19th International Conference, CAiSE 2007, Trondheim, Norway, June 11-15, 2007, Proceedings*, volume 4495 of *Lecture Notes in Computer Science*, pages 574–588. Springer.

[274] Weber, P., Bordbar, B., and Tiňo, P. (2011a). A principled approach to the analysis of process mining algorithms. In *Intelligent Data Engineering and Automated Learning - IDEAL 2011 - 12th International Conference. Proceedings*, volume 6936 of *Lecture Notes in Computer Science*, pages 474–481. Springer.

[275] Weber, P., Bordbar, B., and Tiňo, P. (2011b). Real-time detection of process change using process mining. In *2011 Imperial College Computing Student Workshop, ICCSW 2011. Proceedings*, volume DTR11-9 of *Department of Computing Technical Report*, pages 108–114. Imperial College London.

[276] Weber, P., Tiňo, P., and Bordbar, B. (2012). Process mining in non-stationary environments. In *20th European Symposium on Artificial Neural Networks, ESANN 2012*.

[277] Weidlich, M., Ziekow, H., Mendling, J., Günther, O., Weske, M., and Desai, N. (2011). Event-based monitoring of process execution violations. In *Business Process Management - 9th International Conference, BPM 2011. Proceedings*, volume 6896 of *Lecture Notes in Computer Science*, pages 182–198. Springer.

[278] Weijters, A. J. M. M., van der Aalst, W. M. P., and Alves de Medeiros, A. K. (2006). Process mining with the heuristics miner-algorithm. volume WP 166 of *BETA Working Paper Series*. Eindhoven University of Technology.

[279] Wen, L., van der Aalst, W. M. P., Wang, J., and Sun, J. (2007). Mining process models with non-free-choice constructs. *Data Min. Knowl. Discov.*, 15(2):145–180.

[280] Wen, L., Wang, J., van der Aalst, W. M. P., Huang, B., and Sun, J. (2009). A novel approach for process mining based on event types. *J. Intell. Inf. Syst.*, 32(2):163–190.

[281] Westergaard, M. and Maggi, F. M. (2011). Declare: A Tool Suite for Declarative Workflow Modeling and Enactment. In *Proceedings of the Demo Track of the Nineth Conference on Business Process Management 2011*, volume 820 of *CEUR Workshop Proceedings*.

[282] Wetzstein, B., Ma, Z., and Leymann, F. (2008). Towards measuring key performance indicators of semantic business processes. In *Business Information Systems, 11th International Conference, BIS 2008. Proceedings*, volume 7 of *Lecture Notes in Business Information Processing*, pages 227–238. Springer.

[283] Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C., and Bruel, J. (2009). RELAX: incorporating uncertainty into the specification of self-adaptive systems. In *RE 2009, 17th IEEE International Requirements Engineering Conference*, pages 79–88. IEEE Computer Society.

[284] Widmer, G. and Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101.

[285] Winkler, U., Gilani, W., and Marshall, A. (2012). Business driven BCM SLA translation for service oriented systems. In *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance - 16th International GI/ITG Conference, MMB & DFT 2012. Proceedings*, volume 7201 of *Lecture Notes in Computer Science*, pages 206–220. Springer.

[286] Winkler, U., Gilani, W., and Redlich, D. (2013). Model-based business continuity management. US Patent 8,457,996.

[287] Woods, D. and Word, J. (2004). *SAP Netweaver for Dummies*. Wiley Hoboken, Hoboken, New Jersey.

[288] Workflow Management Coalition (WfMC) (2008). Business Process Analytics Format (BPAF) - Draft Specification. Document Number TC-1015, 2.0. http://www.bpm-research.com/wp-content/uploads/2009/02/2009-02-20-wfmc-tc-1015-business-process-analytics-format-r1.pdf, (accessed April 06, 2015).

[289] Workflow Management Coalition (WfMC) (2012). XML Process Definition Language (XPDL) 2.2. http://www.xpdl.org/, (accessed Dec. 12, 2014).

[290] Wynn, M. T., Dumas, M., Fidge, C. J., ter Hofstede, A. H. M., and van der Aalst, W. M. P. (2007). Business process simulation for operational decision support. In *Business Process Management Workshops, BPM 2007 International Workshops, BPI, BPD, CBP, ProHealth, RefMod, semantics4ws, Revised Selected Papers*, volume 4928 of *Lecture Notes in Computer Science*, pages 66–77. Springer.

[291] zur Muehlen, M. (2007). Business Process Management Standards Tutorial. Tutorial at the 5th International Conference on Business Process Management.

[292] zur Muehlen, M. and Shapiro, R. (2015). Business process analytics. In *Handbook on Business Process Management 2, Strategic Alignment, Governance, People and Culture, 2nd Ed.*, International Handbooks on Information Systems, pages 243–263. Springer.

[293] zur Muehlen, M. and Swenson, K. D. (2010). BPAF: A standard for the interchange of process analytics data. In *Business Process Management Workshops - BPM 2010 International Workshops and Education Track, Revised Selected Papers*, volume 66 of *Lecture Notes in Business Information Processing*, pages 170–181. Springer.