Wil van der Aalst

Process Mining

Data Science in Action

Second Edition

Springer

Wil van der Aalst

Process Mining

Data Science in Action

Second Edition

Springer

Wil van der Aalst
Department of Mathematics and Computer Science
Eindhoven University of Technology
Eindhoven, The Netherlands
url: http://www.vdaalst.com

ISBN 978-3-662-49850-7 ISBN 978-3-662-49851-4 (eBook)
DOI 10.1007/978-3-662-49851-4

Library of Congress Control Number: 2016938641

Springer Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Thanks to Karin for understanding that science is more rewarding than running errands

Thanks to all people that contributed to ProM; the fruits of their efforts demonstrate that sharing a common goal is more meaningful than “cashing in the next publon”[1]

In remembrance of Gerry Straatman-Beelen (1932–2010)

[1] Publon = smallest publishable unit.

Preface

The interest in *data science* is rapidly growing. Many consider data science as *the* profession of the future. Just like computer science emerged as a discipline in the 1970s, we now witness the rapid creation of research centers and bachelor/master programs in data science. The hype related to Big Data and predictive analytics illustrates this. Data (“Big” or “small”) are essential for people and organizations and their importance will only increase. However, it is not sufficient to focus on data storage and data analysis. A data scientist also needs to relate data to operational processes and be able to ask the right questions. This requires an understanding of end-to-end processes. *Process mining* bridges the gap between traditional model-based process analysis (e.g., simulation and other business process management techniques) and data-centric analysis techniques such as machine learning and data mining. Process mining provides a new means to improve processes in a variety of application domains. The omnipresence of event data combined with process mining allows organizations to diagnose problems based on facts rather than fiction.

Although traditional Business Process Management (BPM) and Business Intelligence (BI) technologies received lots of attention, they did not live up to the expectations raised by academics, consultants, and software vendors. Probably, the same will happen to most of the Big Data technologies vigorously promoted today. The goal should be to improve the operational processes themselves rather than the artifacts (models, data, and systems) they use. As will be demonstrated in this book, there are novel ways to put “data science in action” and improve processes based on the data they generate.

Process mining is an emerging discipline providing comprehensive sets of tools to provide fact-based insights and to support process improvements. This new discipline builds on process model-driven approaches and data mining. However, process mining is much more than an amalgamation of existing approaches. For example, existing data mining techniques are too data-centric to provide a comprehensive understanding of the end-to-end processes in an organization. BI tools focus on simple dashboards and reporting rather than clear-cut business process insights. BPM suites heavily rely on experts modeling idealized to-be processes and do not help the stakeholders to understand the as-is processes.

This book presents a range of process mining techniques that help organizations to uncover their actual business processes. Process mining is not limited to process discovery. By tightly coupling event data and process models, it is possible to check conformance, detect deviations, predict delays, support decision making, and recommend process redesigns. Process mining breathes life into otherwise static process models and puts today’s massive data volumes in a process context. Hence, managements trends related to process improvement (e.g., Six Sigma, TQM, CPI, and CPM) and compliance (SOX, Basel II, etc.) can benefit from process mining.

Process mining, as described in this book, emerged in the last decade [156, 160]. However, the roots date back about half a century. For example, Anil Nerode presented an approach to synthesize finite-state machines from example traces in 1958 [108], Carl Adam Petri introduced the first modeling language adequately capturing concurrency in 1962 [111], and Mark Gold was the first to systematically explore different notions of learnability in 1967 [61]. When data mining started to flourish in the 1990s, little attention was given to processes. Moreover, only recently event logs have become omnipresent thus enabling end-to-end process discovery. Since the first survey on process mining in 2003 [156], progress has been spectacular. Process mining techniques have become mature and supported by various tools. Moreover, whereas initially the primary focus was on process discovery, the process mining spectrum has broadened markedly. For instance, conformance checking, multi-perspective process mining, and operational support have become integral parts of ProM, one of the leading process mining tools.

The book provides a comprehensive overview of the state-of-the-art in process mining. It is intended as an introduction to the topic for practitioners, students, and academics. On the one hand, the book is accessible for people that are new to the topic. On the other hand, the book does not avoid explaining important concepts on a rigorous manner. The book aims to be self-contained while covering the entire process mining spectrum from process discovery to operational support. Therefore, it also serves as a reference handbook for people dealing with BPM or BI on a day-to-day basis.

The first edition of this book appeared in 2011 under the title “Process Mining: Discovery, Conformance and Enhancement of Business Processes” [140]. Given the rapid developments in process mining, there was a clear need for an updated version. The original book has been extended in several ways. First of all, process mining has been put into the broader context of *data science* (see the new Chap. 1). This explains the new subtitle “Data Science in Action”. There is an urgent need for data scientists able to help organizations improve their operational processes. Therefore, the new edition of the book positions process mining in this broader context and relates it to statistics, data mining, Big Data, etc. Second, there has been significant progress in process discovery in recent years. This is exemplified by the family of *inductive mining* techniques that can handle large incomplete event logs with infrequent behavior, but still provide formal guarantees. The basic elements of inductive mining (Sect. 7.5) and the notion of process trees (Sect. 3.2.8) have been added to this book. Third, the notion of *alignments* has become a key concept to relate observed behavior and modeled behavior. The chapter on conformance checking has been extended to carefully introduce alignments (Sect. 8.3). Moreover, next

to fitness, also quality dimensions like *precision* are now defined. Fourth, a chapter on “process mining in the large” (Chap. 12) has been added to illustrate that process mining can exploit modern infrastructures and that process discovery and conformance checking can be decomposed and distributed. Since the first edition of the book, many new process mining products emerged (often inspired by the open source platform ProM and the previous edition of this book). The chapter on tools (Chap. 11) has been completely rewritten and discusses commercial tools like Celonis Process Mining, Disco, Enterprise Discovery Suite, Interstage Business Process Manager Analytics, Minit, myInvenio, Perceptive Process Mining, QPR ProcessAnalyzer, Rialto Process, SNP Business Process Analysis, and webMethods Process Performance Manager (next to open-source initiatives like ProM and RapidProM). Finally, pointers to recent literature have been added and a new section of data quality has been added (Sect. 5.4). These changes justify a revised edition of the book.

The reader can immediately put process mining into practice due to the applicability of the techniques, the availability of (open-source) process mining software, and the abundance of event data in today’s information systems. I sincerely hope that you enjoy reading this book and start using some of the amazing process mining techniques available today.

Eindhoven, The Netherlands — Wil van der Aalst
January 2016

Acknowledgements

Many individuals and organizations contributed to the techniques and tools described in this book. Therefore, it is a pleasure to acknowledge their support, efforts, and contributions.

All of this started in 1999 with a research project named “Process Design by Discovery: Harvesting Workflow Knowledge from Ad-hoc Executions” initiated by Ton Weijters and myself. At that time, I was still working as a visiting professor at the University of Colorado in Boulder. However, the research school BETA had encouraged me to start collaborating with existing staff in my new research group at TU/e (Eindhoven University of Technology). After talking to Ton it was clear that we could benefit from combining his knowledge of machine learning with my knowledge of workflow management and Petri nets. Process mining (at that time we called it workflow mining) was the obvious topic for which we could combine our expertise. This was the start of a very successful collaboration. Thanks Ton!

Since the turn of the century, many PhD students have been working on the topic: Arya Adriansyah, Ana Karla Alves de Medeiros, Alfredo Bolt Iriondo, R.P. Jagadeesh Chandra (JC) Bose, Carmen Bratosin, Joos Buijs, Alok Dixit, Boudewijn van Dongen, Maikel van Eck, Robert Engel, Eduardo González Lopéz de Murillas, Christian Günther, Bart Hompes, Anna Kalenkova, Marie Koorneef, Maikel Leemans, Sander Leemans, Guangming Li, Cong Liu, Xixi Lu, Felix Mannhardt, Ronny Mans, Laura Maruster, Alexey Mitsyuk, Richard Müller, Jorge Munoz-Gama, Joyce Nakatumba, Maja Pesic, Elham Ramezani, Anne Rozinat, Alifah Syamsiyah, Helen Schonenberg, Dennis Schunselaar, Minseok Song, Niek Tax, and Bas van Zelst. I am extremely grateful for their efforts.

Ana Karla Alves de Medeiros was the first PhD student to work on the topic under my supervision (genetic process mining). She did a wonderful job; her thesis on genetic process mining was awarded with the prestigious ASML 2007 Promotion Prize and was selected as the best thesis by the KNAW research school BETA. Also Boudewijn van Dongen has been involved in the development of ProM right from the start. As a Master student he already developed the process mining tool EMiT, i.e., the predecessor of ProM. He turned out to be a brilliant PhD student and developed a variety of process mining techniques. Eric Verbeek did a PhD on work-

flow verification, but over time he got more and more involved in process mining research and the development of ProM. Many people underestimate the importance of a scientific programmer like Eric. Tool development and continuity are essential for scientific progress! Boudewijn and Eric have been the driving force behind ProM and their contributions have been crucial for process mining research at TU/e. Moreover, they are always willing to help others. Thanks guys!

Christian Günther and Anne Rozinat joined the team in 2005. Their contributions have been of crucial importance for extending the scope of process mining and lifting the ambition level. Christian managed to make ProM look beautiful while significantly improving its performance. Moreover, his Fuzzy miner facilitated dealing with Spaghetti processes. Anne managed to widen the process mining spectrum by adding conformance checking and multi-perspective process mining to ProM. It is great that they succeeded in founding a process mining company (Fluxicon). Anne and Christian are great process mining ambassadors and build software that people can and also want to use. Another person crucial for the development of ProM is Peter van den Brand. He set up the initial framework and played an important role in the development of the architecture of ProM 6. Based on his experience with ProM, he set up a process mining company (Futura Process Intelligence) that joined forces with Pallas Athena which, in turn, was taken over by Lexmark’s Perceptive Software. It is great to work with people like Peter, Christian, and Anne; they are essential for turning research results into commercial products (although I am still waiting for the sports cars they promised . . .).

Next to Boudewijn, Eric, and the PhDs mentioned, the current “process mining team” at TU/e consists of Joos Buijs, Dirk Fahland, Massimiliano de Leoni, Hajo Reijers, Natalia Sidorova, Patrick Mukala, Nour Assy, Farideh Heidari, and—of course—Ine van der Ligt, our secretary.

Academics from various universities contributed to ProM and supported our process mining research. We are grateful to the Technical University of Lisbon, Katholieke Universiteit Leuven, Universitat Politècnica de Catalunya, Universität Paderborn, University of Rostock, Humboldt-Universität zu Berlin, University of Calabria, Queensland University of Technology, Tsinghua University, Universität Innsbruck, Ulsan National Institute of Science and Technology, Università di Bologna, Zhejiang University, Vienna University of Technology, Universität Ulm, Open University, Jilin University, National Research University Higher School of Economics, Free University of Bozen-Bolzano, University of Tartu, Pontificia Universidad Católica de Chile, University of Vienna, Pontificia Universidade Católica do Paraná, Technion, VU University Amsterdam, Hasso-Plattner-Institut, University of Freiburg, Vienna University of Economics and Business, University of Haifa, University of Naples Federico II, University of Padua, and University of Nancy for their help. I would also like to thank the members of the IEEE Task Force on Process Mining for promoting the topic. We are grateful to all other organizations that supported process mining research at TU/e: NWO, STW, EU, IOP, LOIS, BETA, SIKS, Stichting EIT Informatica Onderwijs, Pallas Athena, IBM, LaQuSo, Philips Healthcare, Philips Research, Vanderlande, BrandLoyalty, ESI, Jacquard, Nuffic, BPM Usergroup, and WWTF. Special thanks go to Pallas Athena and Fluxicon

for promoting the topic of process mining and their collaboration in a variety of projects. Over 150 organizations provided event logs that helped us to improve our process mining techniques. Here, I would like to explicitly mention the AMC hospital, Philips Healthcare, ASML, Ricoh, Vestia, Catharina hospital, Thales, Océ, Rijkswaterstaat, Heusden, Harderwijk, Deloitte, and all organizations involved in the SUPER, ACSI, PoSecCo, and CoSeLoG projects. We are grateful for allowing us to use their data and for providing feedback.

Since 2013 I am also serving as the scientific director of the *Data Science Center Eindhoven* (DSC/e). This is a great initiative. Research and education in data science are of growing importance, and there is a natural fit with process mining. I am grateful for the support from people like Emile Aarts, Maurice Groten, Joos Buijs, Jack van Wijk, and many others. They helped to create and develop DSC/e.

It is impossible to name all of the individuals that contributed to ProM or helped to advance process mining. Peter van den Brand, Boudewijn van Dongen, Dirk Fahland, Christian Günther, Sander Leemans, Xixi Lu, Massimiliano de Leoni, Felix Mannhardt, Eric Verbeek, Michael Westergaard, and many others contributed to the current version of ProM. Moreover, besides the people mentioned earlier, I would like to thank Han van der Aa, Rafael Accorsi, Michael Adams, Piet Bakker, Huub de Beer, Tobias Blickle, Seppe vanden Broucke, Andrea Burattin, Riet van Buul, Toon Calders, Diego Calvanese, Jorge Cardoso, Josep Carmona, Alina Chipaila, Jan Claes, Raffaele Conforti, Francisco Curbera, Ernesto Damiani, Marcus Dees, Benoît Depaire, Jörg Desel, John Domingue, Marlon Dumas, Schahram Dustdar, Skip Ellis, Paul Eertink, Dyon Egberts, Dirk Fahland, Diogo Ferreira, Colin Fidge, Walid Gaaloul, Frank van Geffen, Stijn Goedertier, Adela Grando, Gianluigi Greco, Vladimir Gromov, Dolf Grünbauer, Shengnan Guo, Antonella Guzzo, Kees van Hee, Joachim Herbst, Sergio Hernández, Rastislav Hlavac, Arthur ter Hofstede, John Hoogland, Mieke Jans, Theo Janssen, Urszula Jessen, Georgi Jojgov, Ivo de Jong, Ivan Khodyrev, Albert Kisjes, Martin Klenk, Pieter de Kok, Angelique Koopman, Rudolf Kuhn, Thom Langerwerf, Teemu Lehto, Giorgio Leonardi, Jiafei Li, Zheng Liu, Niels Lohmann, Irina Lomazova, Wei Zhe Low, Peter Hornix, Fabrizio Maggi, Paola Mello, Jan Mendling, Frits Minderhoud, Arnold Moleman, Marco Montali, Michael zur Muehlen, Mariska Netjes, Andriy Nikolov, Rudi Niks, Bastian Nominacher, Chun Ouyang, Zbigniew Paszkiewicz, Mykola Pechenizkiy, Carlos Pedrinaci, Anastasiia Pika, Viara Popova, Silvana Quaglini, Manfred Reichert, Remmert Remmerts de Vries, Joel Ribeiro, Jaap Rigter, Stefanie Rinderle-Ma, Alexander Rinke, Andreas Rogge-Solti, Marcello La Rosa, Michael Rosemann, Marcella Rovani, Vladimir Rubin, Nick Russell, Stefania Rusu, Eduardo Portela Santos, Marcos Sepúlveda, Shiva Shabaninejad, Pnina Soffer, Alessandro Sperduti, Christian Stahl, Suriadi Suriadi, Keith Swenson, Nikola Trcka, Kenny van Uden, Irene Vanderfeesten, Jan Vanthienen, Rob Vanwersch, George Varvaressos, Marc Verdonk, Sicco Verwer, Jan Vogelaar, Hans Vrins, Jianmin Wang, Teun Wagemakers, Barbara Weber, Jochen De Weerdt, Lijie Wen, Jan Martijn van der Werf, Mathias Weske, Jack van Wijk, Moe Wynn, Bart Ydo, Marco Zapletal, Reng Zeng, and Indre Zliobaite for their support.

Thanks to Springer-Verlag for publishing this book. Ralf Gerstner encouraged me to write this book and handled things in a truly excellent manner. He also repeatedly triggered me to make a new edition of this book. Thanks Ralf!

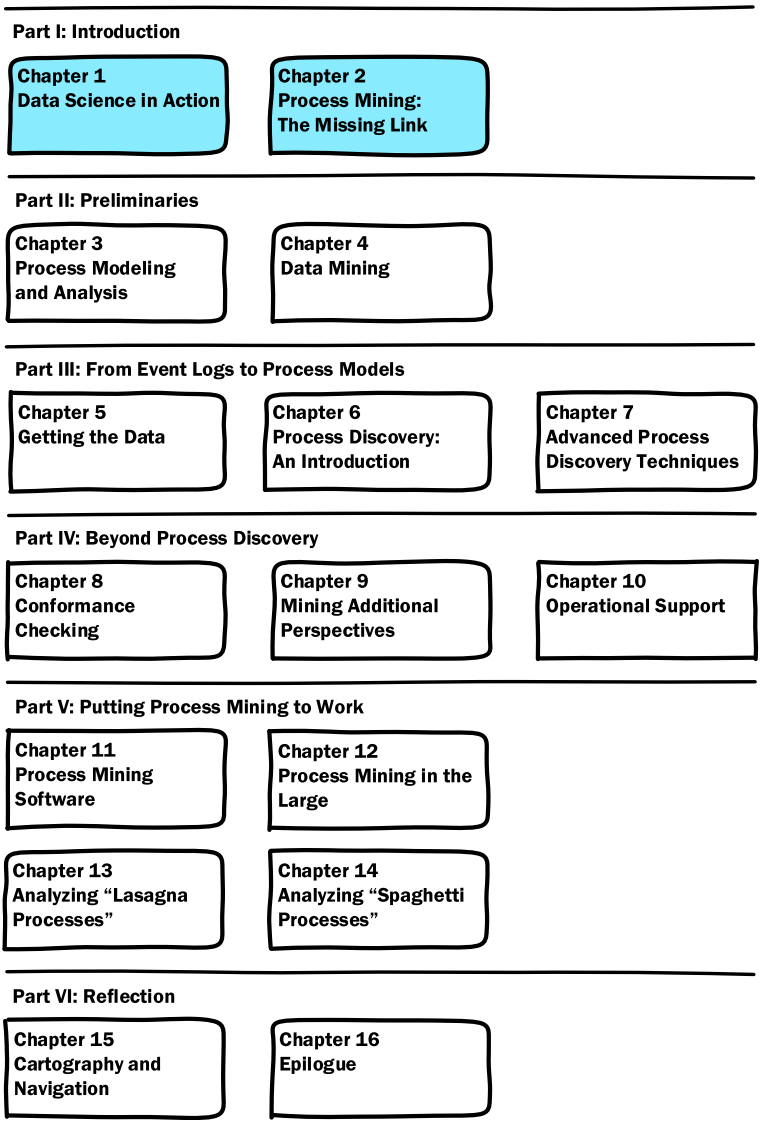
More than 95% of the original book was written in beautiful Schleiden. Despite my sabbatical, there were many other tasks competing for attention. Thanks to my weekly visits to Schleiden (without Internet access!), it was possible to write the first edition of this book in a three month period. The excellent coffee of Serafin helped when proofreading the individual chapters, the scenery did the rest.

As always, acknowledgements end with thanking the people most precious. Lion’s share of credits should go to Karin, Anne, Willem, Sjaak, and Loes. They often had to manage without me under difficult circumstances. Without their continuing support, this book would have taken ages.

Eindhoven, The Netherlands Wil van der Aalst
January 2016

Contents

Part I
Introduction



The goal of process mining is to turn event data into insights and actions. Process mining is an integral part of data science, fueled by the availability of data and the desire to improve processes. Part I sets the scene for the more technical chapters on process modeling, data mining, data extraction, process discovery, conformance checking, performance analysis, and operational support. Chapter 1 starts with an overview of the data science discipline and is used to position process mining. Chapter 2 introduces the basic concepts of process mining.

Chapter 1
Data Science in Action

In recent years, *data science* emerged as a new and important discipline. It can be viewed as an amalgamation of classical disciplines like statistics, data mining, databases, and distributed systems. Existing approaches need to be combined to turn abundantly available data into value for individuals, organizations, and society. Moreover, new challenges have emerged, not just in terms of size (“Big Data”) but also in terms of the questions to be answered. This book focuses on the *analysis of behavior based on event data*. *Process mining* techniques use event data to discover processes, check compliance, analyze bottlenecks, compare process variants, and suggest improvements. In later chapters, we will show that process mining provides powerful tools for today’s data scientist. However, before introducing the main topic of the book, we provide an overview of the data science discipline.

1.1 Internet of Events

As described in [73], society shifted from being predominantly “analog” to “digital” in just a few years. This has had an incredible impact on the way we do business and communicate [99]. Society, organizations, and people are “Always On”. Data are collected *about anything*, *at any time*, and *at any place*. Nowadays, the term “Big Data” is often used to refer the expanding capabilities of information systems and other systems that depend on computing. These developments are well characterized by *Moore’s law*. Gordon Moore, the co-founder of Intel, predicted in 1965 that the number of components in integrated circuits would double every year. During the last 50 years the growth has indeed been exponential, albeit at a slightly slower pace. For example, the number of transistors on integrated circuits has been doubling every two years. Disk capacity, performance of computers per unit cost, the number of pixels per dollar, etc. have been growing at a similar pace. Besides these incredible technological advances, people and organizations depend more and more on computerized devices and information sources on the Internet. The IDC Digital Universe Study of April 2014 confirms again the spectacular growth of data [134].

W. van der Aalst, *Process Mining*, DOI 10.1007/978-3-662-49851-4_1

This study estimates that the amount of digital information (cf. personal computers, digital cameras, servers, sensors) stored in 2014 already exceeded 4 Zettabytes and predicts that the “digital universe” will to grow to 44 Zettabytes in 2020. The IDC study characterizes 44 Zettabytes as “6.6 stacks of iPads from Earth to the Moon”. This illustrates that the long anticipated *data explosion* has become an undeniable reality.

From Bits to Zettabytes
A “bit” is the smallest unit of information possible. One bit has two possible values: 1 (on) and 0 (off). A “byte” is composed of 8 bits and can represent $2^8 = 256$ values. To talk about larger amounts of data, multiples of 1000 are used: 1 Kilobyte (KB) equals 1000 bytes, 1 Megabyte (MB) equals 1000 KB, 1 Gigabyte (GB) equals 1000 MB, 1 Terabyte (TB) equals 1000 GB, 1 Petabyte (PB) equals 1000 TB, 1 Exabyte (EB) equals 1000 PB, and 1 Zettabyte (ZB) equals 1000 EB. Hence, 1 Zettabyte is $10^{21} = 1{,}000{,}000{,}000{,}000{,}000{,}000{,}000$ bytes. Note that here we used the International System of Units (SI) set of unit prefixes, also known as SI prefixes, rather than binary prefixes. If we assume binary prefixes, then 1 Kilobyte is $2^{10} = 1024$ bytes, 1 Megabyte is $2^{20} = 1048576$ bytes, and 1 Zettabyte is $2^{70} \approx 1.18 \times 10^{21}$ bytes.

Most of the data stored in the digital universe is unstructured, and organizations have problems dealing with such large quantities of data. One of the main challenges of today’s organizations is to *extract information and value from data* stored in their information systems.

The importance of information systems is not only reflected by the spectacular growth of data, but also by the role that these systems play in today’s business processes as the digital universe and the physical universe are becoming more and more aligned. For example, the “state of a bank” is mainly determined by the data stored in the bank’s information system. Money has become a predominantly digital entity. When booking a flight over the Internet, a customer is interacting with many organizations (airline, travel agency, bank, and various brokers), often without being aware of it. If the booking is successful, the customer receives an e-ticket. Note that an e-ticket is basically a number, thus illustrating the alignment between the digital and physical universe. When the SAP system of a large manufacturer indicates that a particular product is out of stock, it is impossible to sell or ship the product even when it is available in physical form. Technologies such as RFID (Radio Frequency Identification), GPS (Global Positioning System), and sensor networks will stimulate a further alignment of the digital universe and the physical universe. RFID tags make it possible to track and trace individual items. Also note that more and more devices are being monitored. Already 14 billion devices are connected to the Internet [134]. For example, Philips Healthcare is monitoring its medical equipment (e.g., X-ray machines and CT scanners) all over the world. This helps Philips to

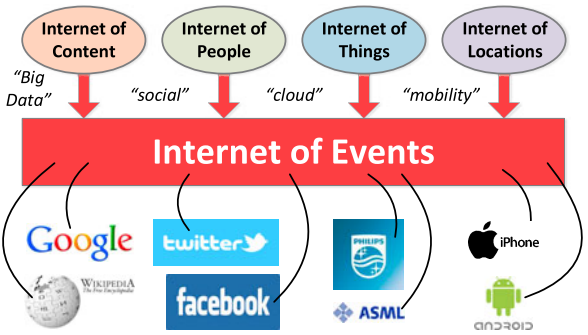


Fig. 1.1 Internet of Events (IoE): Event data are generated from a variety of sources connected to the Internet

understand the needs of customers, test their systems under realistic circumstances, anticipate problems, service systems remotely, and learn from recurring problems. The success of the “App Store” of Apple illustrates that location-awareness combined with a continuous Internet connection enables new ways to pervasively intertwine the digital universe and the physical universe.

The spectacular growth of the digital universe, summarized by the overhyped term “Big Data”, makes it possible to record, derive, and analyze *events*. Events may take place inside a machine (e.g., an X-ray machine, an ATM, or baggage handling system), inside an enterprise information system (e.g., an order placed by a customer or the submission of a tax declaration), inside a hospital (e.g., the analysis of a blood sample), inside a social network (e.g., exchanging e-mails or twitter messages), inside a transportation system (e.g., checking in, buying a ticket, or passing through a toll booth), etc. Events may be “life events”, “machine events”, or “organization events”. The term *Internet of Events* (IoE), coined in [146], refers to all event data available. The IoE is composed of:

- The *Internet of Content* (IoC), i.e., all information created by humans to increase knowledge on particular subjects. The IoC includes traditional web pages, articles, encyclopedia like Wikipedia, YouTube, e-books, newsfeeds, etc.
- The *Internet of People* (IoP), i.e., all data related to social interaction. The IoP includes e-mail, Facebook, Twitter, forums, LinkedIn, etc.
- The *Internet of Things* (IoT), i.e., all physical objects connected to the network. The IoT includes all things that have a unique id and a presence in an Internet-like structure.
- The *Internet of Locations* (IoL) which refers to all data that have a geographical or geospatial dimension. With the uptake of mobile devices (e.g., smartphones) more and more events have location or movement attributes.

Note that the IoC, the IoP, the IoT, and the IoL are overlapping. For example, a place name on a webpage or the location from which a tweet was sent. *Process mining aims to exploit event data in a meaningful way*, for example, to provide insights, identify bottlenecks, anticipate problems, record policy violations, recommend countermeasures, and streamline processes. This explains our focus on event data.

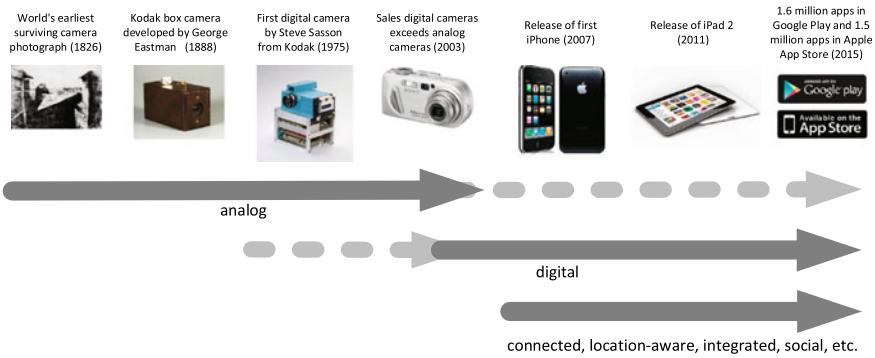


Fig. 1.2 The transition from analog to digital dramatically changed the way we create and share photos. This is one of the factors contributing to the rapid expansion of the Internet of Events (IoE)

To illustrate the above developments, let us consider the development of photography over time (see Fig. 1.2). Photography emerged at the beginning of the 19th century. Around 1800, Thomas Wedgwood attempted to capture the image in a camera obscura by means of a light-sensitive substance. The earliest remaining photo dates from 1826. Towards the end of the 19th century, photographic techniques matured. George Eastman founded Kodak around 1890 and produced “The Kodak” box camera that was sold for $25, thus making photography accessible for a larger group of people. The company witnessed the rapid growth of photography while competing with companies like Fujifilm. In 1976, Kodak was responsible for 90% of film sales and 85% of camera sales in the United States [57]. Kodak developed the first digital camera in 1975, i.e., at the peak of its success. The Kodak digital camera had the size of a toaster and a CCD image sensor that only allowed for 0.01 megapixel black and white pictures. It marked the beginning of digital photography, but also the decline of Kodak. Kodak was unable to adapt to the market of digital photography. Competitors like Sony, Canon, and Nikon better adapted to the rapid transition from analog to digital. In 2003, the sales of digital cameras exceeded the sales of traditional cameras for the first time. Today, the market for analog photography is virtually non-existent. Soon after their introduction, smartphones with built-in cameras overtook dedicated cameras. The first iPad having a camera (iPad 2) was presented on March 2nd, 2011 by Steve Jobs. Today, the sales of tablet-like devices like the iPad exceed the sales of traditional PCs (desktops and laptops). As a result of these developments, most photos are made using mobile phones and tablets. The remarkable transition from analog to digital photography has had an impact that goes far beyond the photos themselves. Today, photos have GPS coordinates allowing for localization. Photos can be shared online (e.g., Flickr, Instagram, Facebook, and Twitter) and changed the way we communicate and socialize (see the uptake of the term “selfie”). Smartphone apps can detect eye cancer, melanoma, and other diseases by analyzing photos. A photo created using a smartphone may generate a wide range of events (e.g., sharing) having data attributes (e.g., location) that reach far beyond the actual image. As illustrated by

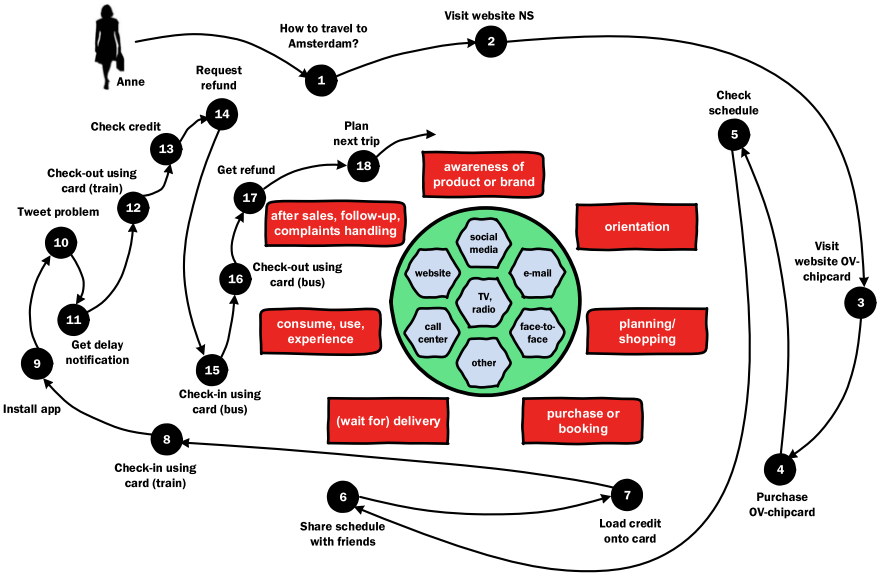


Fig. 1.3 An example of a customer journey illustrating the many (digital) touchpoints generating events that allow us to understand and serve customers better

Fig. 1.2, developments in photography accelerated since the first digital camera, and the transition from analog to digital photography contributed significantly to the growth of the Internet of Events (IoE). Digitalization resulted not just in content (e.g., photos) but also in new ways to capture “events” showing what is really happening.

Let us now consider another development: the digitization[1] of the *customer journey* where customers interact in multiple novel ways with organizations [36]. In the digital era, there are many *touchpoints* using different media. The center of Fig. 1.3 shows the different media: social media, e-mails, websites, face-to-face contacts, call-centers, etc. Although there are significant differences between the wide variety communication channels, “content” tends to become less dependent on the media used (phone, laptop, etc.). Smartphones and iPads can make photographs, computers can be used to make phone calls (through Skype or FaceTime), and customer complaints can be expressed via a website or call-center. Different devices and services co-exist in an integrated ecosystem. Consider, for example, the capturing, managing, publication, viewing, and sharing of photos using digital cameras, mobile phones, computers, websites, media-players, printers, televisions, interactive whiteboards, etc.

[1] Some distinguish between the terms digitization and digitalization where *digitization* is the process of converting an analogue reality into digital bits and *digitalization* refers to the way in which social life and businesses are impacted by digital communication infrastructures. In this book, we do not use this distinction as both are too much intertwined.

Figure 1.3 distinguishes seven stages for an archetypal customer journey:

1. *Awareness of product or brand.* The customer needs to be aware of the product and/or brand to start a customer journey. For example, a customer that does not know about the existence of air purifiers will not consider buying one. (An air purifier removes contaminants from the air in a room to fight allergies, asthmatics, or tobacco smoke.)
2. *Orientation.* During the second stage, the customer is interested in a product, possibly of a particular brand. For example, the customer searches for the differences between air purifiers, e.g., there are devices that use thermodynamic sterilization, ultraviolet germicidal irradiation, HEPA filters, etc.
3. *Planning/shopping.* After the orientation phase the customer many decide to purchase a product or service. This requires planning and/or shopping, e.g., browsing websites for the best offer.
4. *Purchase or booking.* If the customer is satisfied with a particular offering, the product is bought or the service (e.g., flight or hotel) is booked.
5. *(Wait for) delivery.* This is the stage after purchasing the product or booking the service, but before the actual delivery. For example, the air purifier that was purchased is unexpectedly out of stock, resulting in a long delivery time and an unhappy customer. Events like this are an integral part of the customer journey.
6. *Consume, use, experience.* At the sixth stage, the product or service is used. For example, the air purifier arrived and is used on a daily basis. While using the product or service, a multitude of events may be generated. For example, some air purifiers are connected to the Internet measuring the air quality. The user can control the purifier via an app and monitor the air quality remotely. The recorded event data can be used to understand the actual use of the product by the customer.
7. *After sales, follow-up, complaints handling.* This is the stage that follows the actual use of the product or service. For example, the customer may want to return the air purifier because it is broken or does not deliver the performance expected. At this seventh stage, new add-on products may be offered (e.g., air filters).

Given a particular product or organization, many customer journeys are possible. The customer journey is definitely *not* a linear process. Stages may be skipped and revisited. Moreover, customers may use many products of the same brand leading to an overall customer experience influencing future purchase decisions.

Figure 1.3 shows one particular customer journey to illustrate the different touchpoints potentially providing lots of event data for analysis. Consider a teenager (let us call her Anne) that wants to make a trip from Eindhoven Central Station to Amsterdam to visit the Van Gogh Museum. Anne first explores different ways to travel to Amsterdam (1) followed by a visit to the website of NS (Dutch railroad company) (2). Anne finds out that she needs to buy a so-called “OV-chipcard”. Such a card gives access to a contactless smartcard system used for all public transport in the Netherlands. Using the card Anne can check-in at the start of a trip and check-out at the end of trip. After vising the OV-chipcard website (3), Anne purchases

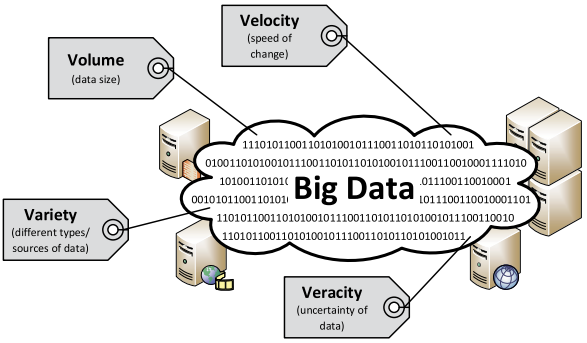


Fig. 1.4 The four V’s of Big Data: Volume, Velocity, Variety, and Veracity

the OV-chipcard from a machine in the train station (4), and checks the schedule (5) using her mobile phone. She shares the selected schedule with her friends (6). Before checking in using the card (8), she first loads 100 euro credit onto her OV-chipcard (7). While traveling she installs the NS app obtained from iTunes (9). Due to a broken cable, the train gets a 90 minute delay. Anne tweets about the problem while mentioning @NS_online to express her disappointment (10). A bit later, she gets a push message from her newly installed app (11). Customers build expectations based on experiences, and Anne is clearly not happy. Due to the digitization of the customer journey, such negative sentiments can be detected and acted upon. Finally, Anne reaches Amsterdam Central Station and checks out (12). Anne checks her credit on the card using a machine (13) and requests a refund using the app on her mobile phone (14). She takes the bus to the Van Gogh Museum. When entering the bus she checks in (15) and checks out (16) when exiting. A few days later she gets the requested refund (17) and starts planning her next trip (18).

During Anne’s journey many events were recorded. It is easy to relate all events involving the OV-chipcard. However, some of the other events may be difficult to relate to Anne. This complicates analysis. *Event correlation*, i.e., establishing relationships between events, is one of the key challenges in data science.

The seven customer journey stages in Fig. 1.3 illustrate that the journey does not end after the 4th stage (purchase or booking). The classical “funnel-oriented” view towards purchasing a product is too restrictive. The availability of customer data from all seven stages helps shifting attention from sales to loyalty.

The development of photography and the many digital touchpoints in today’s customer journey exemplify the growing availability of event data. Although data science is definitely not limited to Big Data, the dimensions of data are rapidly changing resulting in new challenges. It is fashionable to list challenges starting with the letter ‘V’. Figure 1.4 lists the “four V’s of Big Data”: *Volume*, *Velocity*, *Variety*, and *Veracity*. The first ‘V’ (Volume) refers to the incredible scale of some data sources. For example, Facebook has over 1 billion active users and stores hundreds of petabytes of user data. The second ‘V’ (Velocity) refers to the frequency of incoming data that need to be processed. It may be impossible to store all data or the data may change so quickly that traditional batch processing approaches cannot cope with high-velocity streams of data. The third ‘V’ (Variety) refers to the differ-

ent types of data coming from multiple sources. Structured data may be augmented by unstructured data (e.g., free text, audio, and video). Moreover, to derive maximal value, data from different sources needs combined. As mentioned before, the correlation of data is often a major challenge. The fourth ‘V’ (Veracity) refers to the trustworthiness of the data. Sensor data may be uncertain, multiple users may use the same account, tweets may be generated by software rather than people, etc.

Already in 2001, Doug Laney wrote a report introducing the first three V’s [87]. Later the fourth ‘V’ (Veracity) was added. Next to the basic four V’s of Big Data shown in Fig. 1.4, many authors and organizations proposed additional V’s: Variability, Visualization, Value, Venue, Validity, etc. However, there seems to be a consensus that *Volume*, *Velocity*, *Variety*, and *Veracity* are the key characteristics.

Later in this book we will focus exclusively on event data. However, these are an integral part of any Big Data discussion. Input for process mining is an event log which can be seen as a particular view on the event data available. For example, an event log may contain all events related to a subset of customers and used to build a customer journey map.

1.2 Data Scientist

Fueled by the developments just described, *Data science* emerged as a new discipline in recent years. Many definitions have been suggested [48, 112]. For this book, we propose the following definition:

> ***Data science is an interdisciplinary field aiming to turn data into real value. Data may be structured or unstructured, big or small, static or streaming. Value may be provided in the form of predictions, automated decisions, models learned from data, or any type of data visualization delivering insights. Data science includes data extraction, data preparation, data exploration, data transformation, storage and retrieval, computing infrastructures, various types of mining and learning, presentation of explanations and predictions, and the exploitation of results taking into account ethical, social, legal, and business aspects.***

The above definition implies that data science is broader than applied statistics and data mining. *Data scientists* assist organizations in turning data into value. A data scientist can answer a variety of data-driven questions. These can be grouped into the following four main categories [146]:

- (Reporting) *What happened?*
- (Diagnosis) *Why did it happen?*
- (Prediction) *What will happen?*
- (Recommendation) *What is the best that can happen?*

The definition of data science given is quite broad. Some consider data science as just a fancy term for *statistics*. Clearly, data science has its roots in statistics,

a discipline that developed over four centuries. John Graunt (1620–1674) started to study London's death records around 1660. Based on this he was able to predict the life expectancy of a person at a particular age. Francis Galton (1822–1911) introduced statistical concepts like regression and correlation at the end of the 19th century. Although data science can be seen as a continuation of statistics, the majority of statisticians did not contribute much to recent progress in data science. Most statisticians focused on theoretical results rather than real-world analysis problems. The computational aspects, which are critical for larger data sets, are typically ignored by statisticians. The focus is on generative modeling rather than prediction and dealing with practical challenges related to data quality and size. When the data mining community realized major breakthroughs in the discovery of patterns and relationships (e.g., efficiently learning decision trees and association rules), most statisticians referred to these discovery practices as "data fishing", "data snooping", and "data dredging" to express their dismay.

A few well-known statisticians criticized their colleagues for ignoring the actual needs and challenges in data analysis. John Tukey (1915–2000), known for his fast Fourier transform algorithm and the box plots, wrote in 1962: "For a long time I have thought I was a statistician, interested in inferences from the particular to the general. But as I have watched mathematical statistics evolve, I have had cause to wonder and to doubt. ... I have come to feel that my central interest is in data analysis, which I take to include, among other things: procedures for analyzing data, techniques for interpreting the results of such procedures, ways of planning the gathering of data to make its analysis easier, more precise or more accurate, and all the machinery and results of (mathematical) statistics which apply to analyzing data." [133]. This text was written over 50 years ago. Also Leo Breiman (1928–2005), another distinguished statistician, wrote in 2001 "This commitment has led to irrelevant theory, questionable conclusions, and has kept statisticians from working on a large range of interesting current problems. Algorithmic modeling, both in theory and practice, has developed rapidly in fields outside statistics." [25]. David Donoho adequately summarizes the 50 year old struggle between old-school statistics and real-life data analysis in [48].

Data science is also closely related to data processing. Turing award winner Peter Naur (1928–2016) used the term "data science" long before it was in vogue. In 1974, Naur wrote: "A basic principle of *data science*, perhaps the most fundamental that may be formulated, can now be stated: The data representation must be chosen with due regard to the transformation to be achieved and the data processing tools available" [107]. Earlier, Peter Naur also defined *datalogy* as the "science of the nature and use of data" and suggested to use this term rather than "computer science". The book from 1974 also has two parts considering "large data": "Part 5—Processes with Large Amounts of Data" and "Part 6—Large Data Systems" [107]. In the book, "large amounts of data" are all data sets that cannot be stored in working memory. The maximum capacity of magnetic disk stores considered in [107] ranges between 1.25 and 250 megabytes. Not only the disks are orders of magnitude smaller than today's disks, also the notion of what is "large/big" has changed dramatically since the early 1970s. Nevertheless, many of the core principles of data processing have remained invariant.

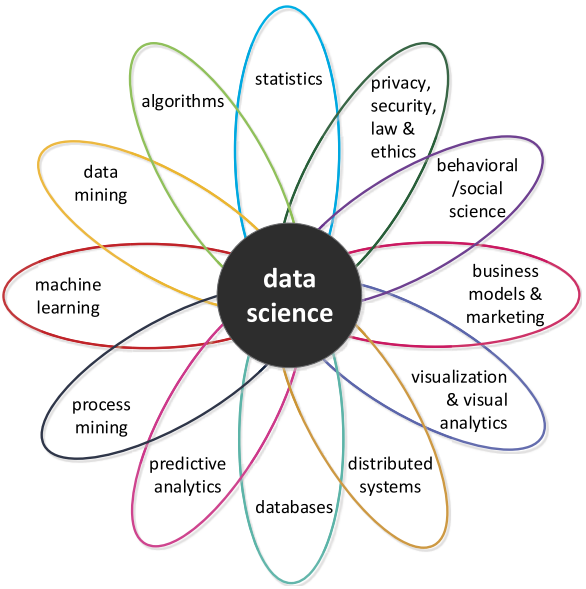


Fig. 1.5 The ingredients contributing to data science

Like data science, computer science had its roots in a number of related areas, including mathematics. Computer science emerged because of the availability of computing resources and the need for computer scientists. Data science is now emerging because of the omnipresence and abundance of data and the need for data scientists that can turn data into value.

Data science is an amalgamation of different partially overlapping (sub)disciplines. Figure 1.5 shows the main ingredients of data science. The diagram should be taken with a grain of salt. The (sub)disciplines are overlapping and varying in size. Moreover, the boundaries are not clear-cut and seem to change over time. Consider, for example, the difference between data mining and machine learning or statistics. Their roots are very different: data mining emerged from the database community, and machine learning emerged from the Artificial Intelligence (AI) community, both quite disconnected from the statistics community. Despite the different roots, the three (sub)disciplines are definitely overlapping.

- *Statistics* can be viewed as the origin of data science. The discipline is typically split into *descriptive* statistics (to summarize sample data using notions like mean, standard deviation, and frequency) and *inferential* statistics (using sample data to estimate characteristics of all data or to test a hypothesis).
- *Algorithms* are crucial in any approach analyzing data. When data sets get larger, the complexity of the algorithms becomes a primary concern. Consider, for example, the Apriori algorithm for finding frequent items sets, the MapReduce approach for parallelizing algorithms, and the PageRank algorithm used by Google search.
- *Data mining* can be defined as “the analysis of (often large) data sets to find unsuspected relationships and to summarize the data in novel ways that are both

understandable and useful to the data owner” [69]. The input data are typically given as a table and the output may be rules, clusters, tree structures, graphs, equations, patterns, etc. Clearly, data mining builds on statistics, databases, and algorithms. Compared to statistics, the focus is on scalability and practical applications.

- *Machine learning* is concerned with the question of how to construct computer programs that automatically improve with experience [102]. The difference between data mining and machine learning is equivocal. The field of machine learning emerged from within Artificial Intelligence (AI) with techniques such as neural networks. Here, we use the term machine learning to refer to algorithms that give computers the capability to learn *without* being explicitly programmed (“learning from experience”). To learn and adapt, a model is built from input data (rather than using fixed routines). The evolving model is used to make data-driven predictions or decisions.
- *Process mining* adds the process perspective to machine learning and data mining. Process mining seeks the confrontation between event data (i.e., observed behavior) and process models (hand-made or discovered automatically). Event data are related to explicit process models, e.g., Petri nets or BPMN models. For example, process models are discovered from event data or event data are replayed on models to analyze compliance and performance.
- *Predictive analytics* is the practice of extracting information from existing data sets in order to determine patterns and predict future outcomes and trends. To generate predictions, existing mining and learning approaches are applied in a business context. Predictive analytics is related to business analytics and business intelligence.
- *Databases* are used to store data. The database discipline forms one of the cornerstones of data science. Database Management (DBM) systems serve two purposes: (i) structuring data so that they can be managed easily and (ii) providing scalability and reliable performance. Using database technology, application programmers do not need to worry about data storage. Until recently, relational databases and SQL (Structured Query Language) were the norm. Due to the growing volume of data, massively distributed databases and so-called NoSQL databases emerged. Moreover, in-memory computing (cf. SAP HANA) can be used to answer questions in real-time. Related is OLAP (Online Analytical Processing) were data are stored in multidimensional cubes facilitating analysis from different points of view.
- *Distributed systems* provide the infrastructure to conduct analysis. A distributed system is composed of interacting components that coordinate their actions to achieve a common goal. Cloud, grid, and utility computing rely on distributed systems. Some analysis tasks are too large or too complex to be performed on a single computer. Such tasks can be split into many smaller tasks that can be performed concurrently on different computing nodes. Scalability may be realized by sharing and/or extending the set of computing nodes.
- *Visualization & visual analytics* are key elements of data science. In the end people need to interpret the results and guide analysis. Automated learning and

mining techniques can be used to extract knowledge from data. However, if there are many "unknown unknowns" (things we don't know we don't know),[2] analysis heavily relies on human judgment and direct interaction with the data. The perception capabilities of the human cognitive system can be exploited by using the right visualizations [178]. Visual analytics, a term coined by Jim Thomas (1946–2010), combines automated analysis techniques with interactive visualizations for an effective understanding, reasoning and decision making on the basis of very large and complex data sets [83].

- *Business models & marketing* also appear in Fig. 1.5 because data science is about turning data into value, including business value. The market capitalization of Facebook in November 2015 was approximately US $300 billion while having approximately 1500 million monthly active users. Hence, the average value of a Facebook user was US $200. At the same time, the average value of a Twitter user was US $55 (market capitalization of approximately US $17 billion with 307 million users). Via the website www.twalue.com one can even compute the value of a particular Twitter account. In November 2015, the author's Twitter account (*@wvdaalst*) was estimated to have a value of US $1002.98. These numbers illustrate the economic value of data and the success of young companies based on new business models. Airbnb (helping people to list, find and rent lodging), Uber (connecting travelers and drivers who use their own cars), and Alibaba (an online business-to-business trading platform) are examples of data-driven companies that are radically changing the hotel, taxi, and trading business. Marketing is also becoming more data-driven (see Sect. 1.1 describing the increase in digital touchpoints during a customer journey). Data scientists should understand how business considerations are driving the analysis of new types of data.
- *Behavioral/social science* appears in Fig. 1.5 because most data are (indirectly) generated by people and analysis results are often used to influence people (e.g., guiding the customer to a product or encouraging a manager to eliminate waste). Behavioral science is the systematic analysis and investigation of human behavior. Social sciences study the processes of a social system and the relationships among individuals within a society. To interpret the results of various types of analytics, it is important to understand human behavior and the social context in which humans and organizations operate. Moreover, analysis results often trigger questions related to coaching and positively influencing people.
- *Privacy, security, law, and ethics* are key ingredients to protect individuals and organizations from "bad" data science practices. Privacy refers to the ability to seclude sensitive information. Privacy often depends on security mechanisms which aim to ensure the confidentiality, integrity and availability of data. Data should be accurate and stored safely, not allowing for unauthorized access. Privacy and security need to be considered carefully in all data science applications. Individuals

[2] On February 12, 2002, when talking about weapons of mass destruction in Iraq, United States Secretary of Defense Donald Rumsfeld used the following classification: (i) "known knowns" (things we know we know), (ii) "known unknowns" (things we know we don't know), and (iii) "unknown unknowns" (things we don't know we don't know).

need to be able to trust the way data are stored and transmitted. Next to concrete privacy and security breaches, there may be ethical notions related to “good” and “bad” conduct. Not all types of analysis possible are morally defendable. For example, mining techniques may favor particular groups (e.g., a decision tree may reveal that it is better to give insurance to middle-aged white males rather than other groups). Moreover, due to a lack of sufficient data, minority groups may be wrongly classified. A data scientist should be aware of such problems and provide safeguards for “irresponsible” forms of data science.

Figure 1.5 shows that data science is quite broad and located at the intersection of existing disciplines. It is difficult to combine all the different skills needed in a single person. Josh Wills, former director of data science at Cloudera, defined a data scientist as “a person who is better at statistics than any software engineer and better at software engineering than any statistician”. It will be a challenge to find and/or educate “unicorn” data scientists able to cover the full spectrum depicted in Fig. 1.5. As a result, ‘unicorn” data scientists are in high demand and extremely valuable for data-driven organizations. As an alternative it is also possible to form multi-disciplinary teams covering the “flower” of Fig. 1.5. In the latter case, it is vital that the team members are able to see the bigger picture and complement each other in terms of skills.

1.3 Bridging the Gap Between Process Science and Data Science

In Fig. 1.5, we listed process mining, the topic of this book, as one of the essential ingredients of data science. Unfortunately, this is not a common view. The process perspective is absent in many Big Data initiatives and data science curricula. We argue that *event data should be used to improve end-to-end processes.* Process mining can be seen as a means to *bridge the gap between data science and process science*. Data science approaches tend to be process agnostic whereas process science approaches tend to be model-driven without considering the “evidence” hidden in the data.

We use the umbrella term “*process science*” to refer to the *broader discipline that combines knowledge from information technology and knowledge from management sciences to improve and run operational processes*. Figure 1.6 shows the ingredients of process science. Just like Fig. 1.5, the diagram should be taken with a grain of salt. The (sub)disciplines mentioned in Fig. 1.6 are also overlapping and varying in size.

- *Stochastics* provides a repertoire of techniques to analyze random processes. The behavior of a process or system is modeled using random variables in order to allow for analysis. Well-known approaches include Markov models, queueing networks/systems, and simulation. These can be used to analyze waiting times, reliability, utilization, etc. in the context stochastic processes.

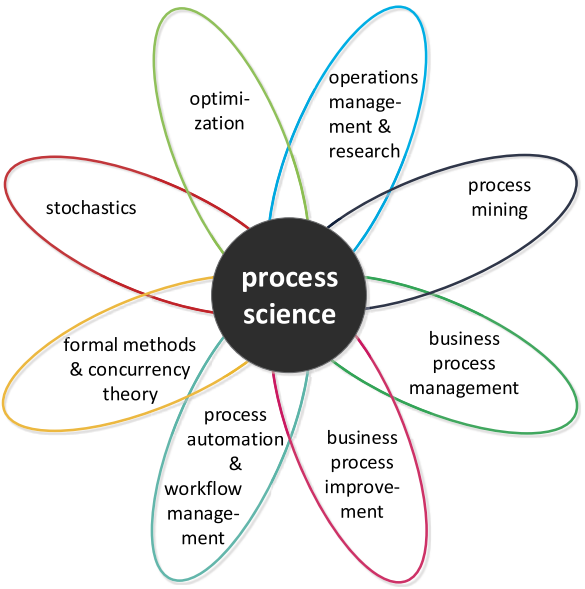


Fig. 1.6 Process science is an umbrella term for the broader discipline that combines knowledge from information technology and knowledge from management sciences to improve and run operational processes

- *Optimization* techniques aim to provide a “best” alternative (e.g., cheapest or fastest) from a large or even infinite set of alternatives. Consider, for example, the following question: Given a list of cities and the distances between each pair of cities, what is a shortest possible route that visits each city exactly once and returns to the origin city? Numerous optimization techniques have been developed to answer such questions as efficient as possible. Well-known approaches include Linear Programming (LP), Integer Linear Programming (ILP), constraint satisfaction, and dynamic programming.
- *Operations management & research* deals with the design, control and management of products, processes, services and supply chains. Operations Research (OR) tends to focus on the analysis of mathematical models. Operations Management (OM) is closer to industrial engineering and business administration.
- *Business process management* is the discipline that combines approaches for the design, execution, control, measurement and optimization of business processes. Business Process Management (BPM) efforts tend to put emphasis on explicit process models (e.g., Petri nets or BPMN models) that describe the control-flow and, optionally, other perspectives (organization, resources, data, functions, etc.) [50, 143, 187].
- *Process mining* is also part of process science. For example, process mining techniques can be used to discover process models from event data. By replaying these data, bottlenecks and the effects of non-compliance can be unveiled. Compared to mainstream BPM approaches the focus is not on process modeling, but on exploiting event data. Sometimes the terms Workflow Mining (WM), Business Process Intelligence (BPI), and Automated Business Process Discovery (ABPD) are used to refer to process-centric data-driven approaches.

- *Business process improvement* is an umbrella term for a variety of approaches aiming at process improvement. Examples are Total Quality Management (TQM), Kaizen, (Lean) Six Sigma, Theory of Constraints (TOC), and Business Process Reengineering (BPR). Note that most of the ingredients in Fig. 1.6 ultimately aim at process improvement, thus making the term business process improvement rather unspecific. One could argue that the whole of process science aims to improve processes.
- *Process automation & workflow management* focuses on the development of information systems supporting operational business processes including the routing and distribution of work. Workflow Management (WFM) systems are model-driven, i.e., a process model suffices to configure the information system and run the process. As a result, a process can be changed by modifying the corresponding process model.
- *Formal methods & concurrency theory* build on the foundations of theoretical computer science, in particular logic calculi, formal languages, automata theory, and program semantics. Formal methods use a range of languages to describe processes. Examples are transition systems, Petri nets, process calculi such as CSP, CCS and π-calculus, temporal logics such as LTL and CTL, and statecharts. Model checkers such as SPIN can be used to verify logical properties such as the absence of deadlocks. Concurrency complicates analysis, but is also essential: In reality parts of a process or system may be executing simultaneously and potentially interacting with each other. Petri nets were the first formalism to model and analyze concurrent processes. Many BPM, WFM, and process mining approaches build upon such formalisms.

As mentioned earlier, Fig. 1.6 should not be taken too seriously. It is merely a characterization of process science and its main ingredients. Note, for example, that stochastics and optimization are partly overlapping (e.g., solving Markov decision processes) and that BPM can be viewed as a continuation or extension of WFM with less emphasis on automation.

Process mining brings together traditional model-based process analysis and data-centric analysis techniques. As shown in Fig. 1.7, *process mining can be viewed as the link between data science and process science*. Process mining seeks the confrontation between event data (i.e., observed behavior) and process models (hand-made or discovered automatically). Mainstream data science approaches tend to be process agnostic. Data mining, statistics and machine learning techniques do not consider end-to-end process models. Process science approaches are process-centric, but often focus on modeling rather than learning from event data. The unique positioning of process mining, as sketched in Fig. 1.7, makes it a powerful tool to exploit the growing availability of data for improving end-to-end processes.

Process mining only recently emerged as a subdiscipline of both data science and process science, but the corresponding techniques can be applied to any type of operational processes (organizations and systems). Example applications include: analyzing treatment processes in hospitals, improving customer service processes in a multinational corporation, understanding the browsing behavior of customers

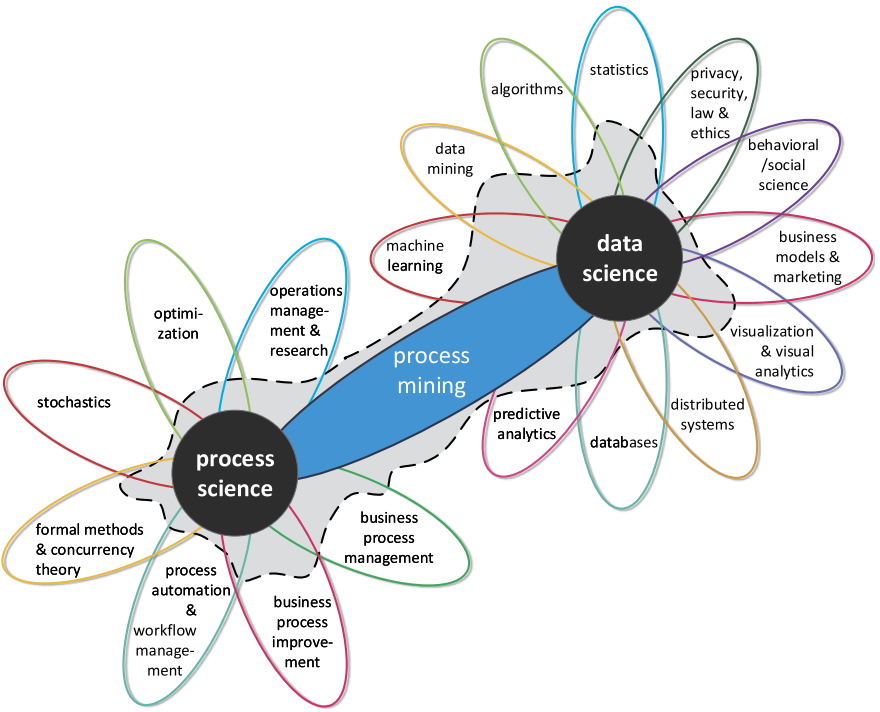


Fig. 1.7 Process mining as the bridge between data science and process science

using a booking site, analyzing failures of a baggage handling system, and improving the user interface of an X-ray machine. What all of these applications have in common is that dynamic behavior needs to be related to process models. Hence, we refer to this as “data science in action”.

Spreadsheets: Dealing with numbers rather than dynamic behavior
Spreadsheet software can be found on most computers, and over the last 25 years many computers have been purchased just to be able to create and use spreadsheets. A spreadsheet is composed of cells organized in rows and columns. Some cells serve as input, other cells have values computed over a collection of other cells (e.g., taking the sum over an array of cells). The expression associated to a cell may use a range of arithmetic operations (add, subtract, multiply, etc.) and predefined functions. For example, Microsoft’s Excel provides hundreds of functions including statistical functions, math and trigonometry functions, financial functions, and logical functions. Most organizations use spreadsheets in financial planning, budgeting, work distribution, etc. Hence, it is interesting to view process mining against the backdrop of this widely used technology.

The first widely used spreadsheet program was VisiCalc (“Visible Calculator”) developed by Dan Bricklin and Bob Frankston, founders of Software Arts (later named VisiCorp). VisiCalc was released in 1979 for the Apple II computer. It is generally considered as Apple II’s “killer application” because numerous organizations purchased the Apple II computer just to be able to use VisiCalc. When Lotus 1-2-3 was launched in 1983, VisiCalc sales dropped dramatically. Lotus 1-2-3 took full advantage of the IBM PC’s capabilities and better supported data handling and charting. What VisiCalc was for the Apple II, Lotus 1-2-3 was for the IBM PC. For the second time, a spreadsheet program generated a tremendous growth in computer sales. People were buying computers in order to run spreadsheet software: A nice example of the “tail” (VisiCalc/Lotus 1-2-3) wagging the “dog” (Apple-II/IBM PC). Lotus 1-2-3 dominated the spreadsheet market until 1992. The dominance ended with the uptake of Microsoft Windows. After decades of spectacular IT-developments, spreadsheet software can still be found on most computers (e.g., Excel is part of Microsoft’s Office) and can be accessed online (e.g., Google Sheets as part of Google Docs).

The situations in which spreadsheets can be used in a meaningful way are almost endless. In short, *spreadsheets can be used to do anything with numbers.* However, spreadsheets are *not* suitable for analyzing event data. One can count frequencies, sums, and the number of events per case using a so-called pivot table, but spreadsheets cannot be used to analyze bottlenecks and deviations (see Fig. 1.8). Consider questions like:

- What are the most frequent paths in my process? Do they change over time?
- What do the cases that take longer than 3 months have in common? Where are the bottlenecks causing these delays?
- Which cases deviate from the reference process? Do these deviations also cause delays?

Obviously, these questions cannot be answered using spreadsheets because the process perspective is completely absent in spreadsheets. Processes *cannot* be captured in numerical data and operations like summation. Process models and concepts such as cases, events, activities, timestamps, and resources need to be treated as first-class citizens during analysis. Data mining tools and spreadsheet programs take as input any tabular data without distinguishing these key concepts. As a result, such tools tend to be process-agnostic. Nevertheless, there is an obvious need for spreadsheet-like technology tailored towards processes and event data.

Where spreadsheets work with *numbers*, process mining starts from *event data* with the aim to analyze processes. Instead of pie charts, bar charts, and tables, results include end-to-end process models, conformance diagnostics, and bottlenecks.

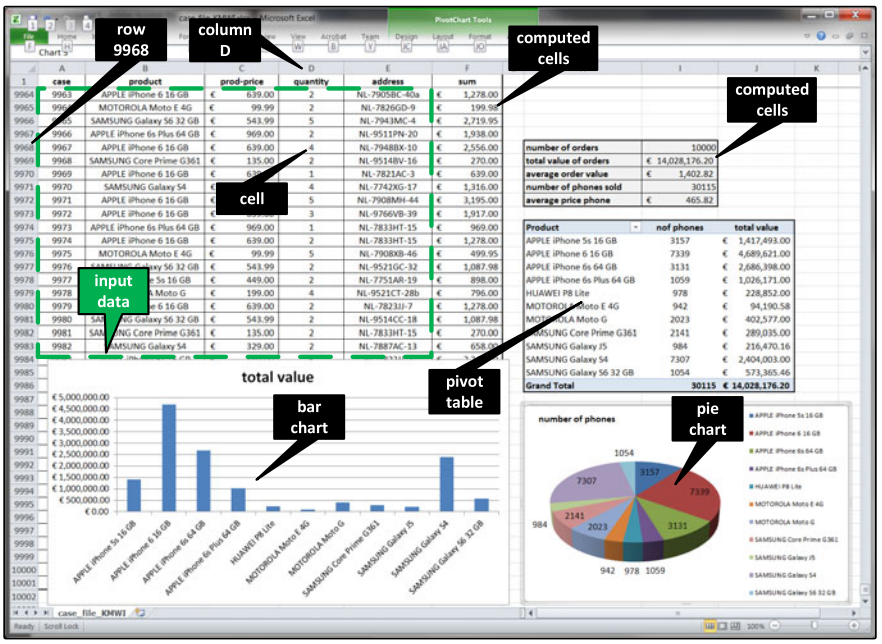


Fig. 1.8 Spreadsheets can be used to do anything with numbers, but have difficulties adequately capturing dynamic behavior

As will be demonstrated in later chapters, the process mining spectrum is quite *broad*. It is not limited to automated process discovery: Process mining can also be used to check compliance, diagnose deviations, pinpoint bottlenecks, improve performance, predict flow times, and recommend actions. Process mining techniques are also *generic*: just like spreadsheet software. Event logs and operational processes can be found everywhere and the analysis techniques are not limited to specific application domains. Just like Excel can be used in finance, production, sales, education, and sports, process mining software can be used in a variety of application domains.

1.4 Outlook

Process mining provides an important bridge between data mining and business process modeling and analysis. Process mining research at TU/e (Eindhoven University of Technology) started in 1999. At that time there was little event data available and the initial process mining techniques were extremely naïve and hence unusable in practice. Over the last decade event data have become readily available and process mining techniques have matured. Moreover, process mining algorithms have been implemented in various academic and commercial systems. Today, there is an active group of researchers working on process mining, and it has become one of the

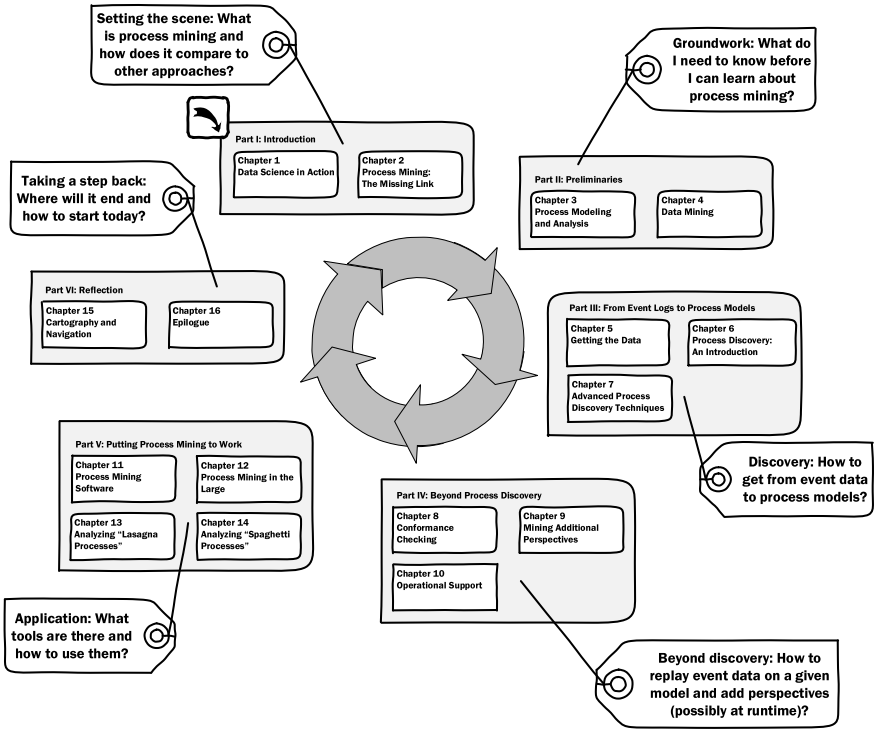


Fig. 1.9 Outline of the book

“hot topics” in BPM research. Moreover, there is a rapidly growing interest from industry in process mining. More and more software vendors started adding process mining functionality to their tools. Our open-source process mining tool ProM is widely used all over the globe and provides an easy starting point for practitioners, students, and academics. These developments are the main motivation for writing this book. There are many books on data mining, business ~~un~~intelligence, process reengineering, and BPM, but these rarely address process mining.

This book aims to provide a comprehensive overview of process mining. The book is intended for business process analysts, business consultants, process managers, graduate students, and BPM researchers. On the one hand, the book avoids delving into unnecessary details. On the other hand, the book does not shy away from formal definitions and technical issues needed to fully understand the essence of process mining. As Einstein said: “Everything should be made as simple as possible, but not one bit simpler”.

Figure 1.9 provides an overview of the book. *Part I* introduces process mining and positions this emerging discipline in the context of data science and process science. *Chap. 2* discusses the role of process models, introduces the notion of event logs, and illustrates the main process mining tasks using a small example.

Part II provides the preliminaries necessary for reading the remainder of the book. *Chap. 3* introduces different process modeling languages and provides an overview of model-based analysis techniques. *Chap. 4* introduces standard data mining techniques such as decision tree learning and association rule learning. Process mining can be seen as a bridge between the preliminaries presented in both chapters.

Part III focuses on one particular process mining task: process discovery. *Chap. 5* discusses the input needed for process mining. The chapter discusses different input formats and issues related to the extraction of event logs from heterogeneous data sources. *Chap. 6* presents the α-algorithm step-by-step in such a way that the reader can understand how it works and see its limitations. This simple algorithm has problems dealing with less structured processes. Nevertheless, it provides a basic introduction into the topic and serves as a “hook” for discussing more advanced algorithms and general issues related to process mining. *Chap. 7* introduces more advanced process discovery approaches. This way the reader gets a good understanding of the state-of-the-art and is guided in selecting suitable techniques.

Part IV moves beyond process discovery, i.e., the focus is no longer on discovering the control-flow. *Chap. 8* presents conformance checking approaches, i.e., techniques to compare and relate event logs and process models. It is shown that conformance can be quantified and that deviations can be diagnosed. *Chap. 9* focuses on other perspectives: the organizational perspective, the case perspective, and the time perspective. *Chap. 10* shows that process mining can also be used to support operational processes at runtime, i.e., while cases are running it is possible to detect violations, make predictions, and provide recommendations.

Part V guides the reader in successfully applying process mining in practice. *Chap. 11* provides an overview of the different process mining tools. Data science is often related to Big Data. The “four V’s of Big Data” (Fig. 1.4) are obviously also relevant for event data and their analysis. *Chap. 12* shows that process mining problems can be decomposed in various ways and many of the techniques can be adapted to provide scalability. The next two chapters are based on the observation that there are essentially two types of processes: “Lasagna processes” and “Spaghetti processes”. Lasagna processes are well-structured and relatively simple. Therefore, process discovery is less interesting, but the techniques presented in Part IV are highly relevant for Lasagna processes. The added value of process mining can be found in conformance checking, detailed performance analysis, and operational support. *Chap. 13* explains how process mining can be applied in such circumstances and provides various real-life examples. Spaghetti processes are less structured. Therefore, the added value of process mining shifts to providing insights and generating ideas for better controlled processes, but advanced techniques such as prediction are less relevant for Spaghetti processes. *Chap. 14* shows how to apply process mining in such less-structured environments.

Part VI takes a step back and reflects on the material presented in the preceding parts. *Chap. 15* provides a broader vision on the topic by comparing process modeling with cartography, and relating BPM systems to navigation systems provided by vendors such as TomTom, Garmin, and Navigon. The goal of this chapter is to provide a refreshing view on process management and reveal the limitations of existing

information systems. *Chap. 16* concludes the book by summarizing improvement opportunities provided by process mining. The chapter also discusses some of the key challenges and provides concrete pointers to start applying the material presented in this book.

Chapter 2
Process Mining: The Missing Link

Information systems are becoming more and more intertwined with the operational processes they support. As discussed in the previous chapter, multitudes of events are recorded by today's information systems. Nevertheless, organizations have problems extracting value from these data. The goal of *process mining* is to use event data to extract process-related information, e.g., to automatically *discover* a process model by observing events recorded by some enterprise system. A small example is used to explain the basic concepts. These concepts will be elaborated in later chapters.

2.1 Limitations of Modeling

Process mining can be viewed as the missing link between *data science* and *process science*, as demonstrated in the previous chapter using Fig. 1.7. Another way to characterize process mining is shown in Fig. 2.1. The diagram shows that process mining starts from *event data* and uses *process models* in various ways, e.g., process models are discovered from event data, serve as reference models, or are used to project bottlenecks on. Figure 2.1 shows that event data and process models can be viewed as “yin and yang” in process mining. Like in Chinese philosophy, we aim for a duality (yin and yang). *Data-driven forces* and *process-centric forces* are viewed as complementary, interconnected, and interdependent in process mining. Figure 2.1 also provides examples of questions that can be answered using process mining. These questions can be grouped into *performance* and *conformance* related questions. Clearly, such questions cannot be answered using a spreadsheet program. We later show concrete examples. However, before introducing process mining using a concrete event log, we first discuss the limitations of modeling when it comes to process analysis and process improvement.

To discuss the limitations of modeling, in particular the use of hand-made models, we briefly introduce *Petri nets* as an example language. A plethora of notations exists to model operational (business) processes (next to Petri nets there are languages like BPMN, UML, and EPCs), some of which will be discussed in the next

W. van der Aalst, *Process Mining*, DOI 10.1007/978-3-662-49851-4_2

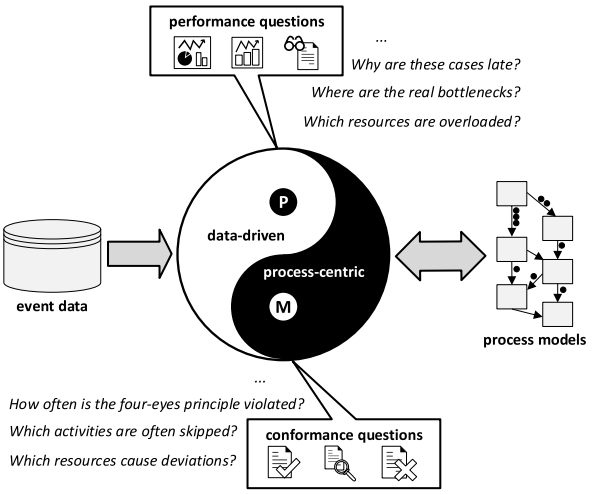


Fig. 2.1 Process mining is both data-driven and process-centric: Using a combination of event data and process models a wide range of conformance and performance questions can be answered

chapter. We refer to all of these as *process models*. The notations mentioned have in common that processes are described in terms of activities (and possibly subprocesses). The ordering of these activities is modeled by describing casual dependencies. Moreover, the process model may also describe temporal properties, specify the creation and use of data, e.g., to model decisions, and stipulate the way that resources interact with the process (e.g., roles, allocation rules, and priorities).

Figure 2.2 shows a process model expressed in terms of a *Petri net* [46]. The model describes the handling of a request for compensation within an airline. Customers may request compensation for various reasons, e.g., a delayed or canceled flight. As Fig. 2.2 shows the process starts by registering the request. This activity is modeled by transition *register request*. Each *transition* is represented by a square. Transitions are connected through *places* that model possible states of the process. Each place is represented by a circle. In a Petri net a transition is *enabled*, i.e., the corresponding activity can occur, if all input places hold a *token*. Transition *register request* has only one input place (*start*) and this place initially contains a token to represent the request for compensation. Hence, the corresponding activity is enabled and can occur. This is also referred to as *firing*. When firing, the transition consumes one token from each of its input places and produces one token for each of its output places. Hence, the firing of transition *register request* results in the removal of the token from input place *start* and the production of two tokens: one for output place *c1* and one for output place *c2*. Tokens are shown as black dots. The configuration of tokens over places—in this case the state of the request—is referred to as *marking*. Figure 2.2 shows the initial marking consisting of one token in place *start*. The marking after firing transition *register request* has two tokens: one in place *c1* and one in place *c2*. After firing transition *register request*, three transitions are enabled. The token in place *c2* enables transition *check ticket*. This transition models an administrative check to see whether the customer is eligible to issue a request. For example, while executing *check ticket* it is verified whether the customer indeed has

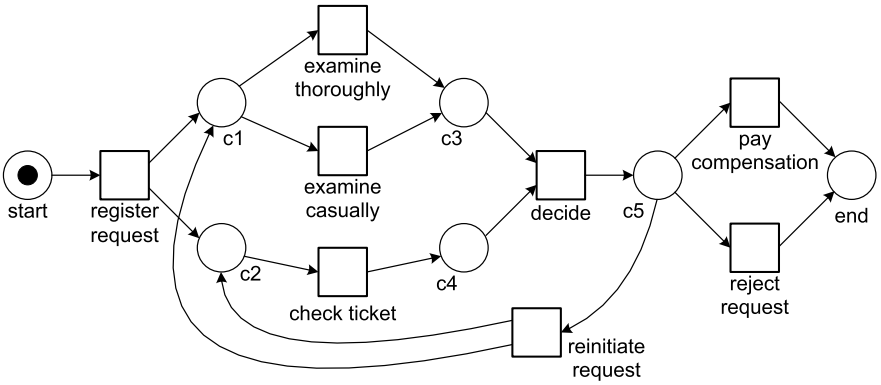


Fig. 2.2 A Petri net modeling the handling of compensation requests

a ticket issued by the airline. In parallel, the token in *c1* enables both *examine thoroughly* and *examine casually*. Firing *examine thoroughly* will remove the token from *c1*, thus disabling *examine casually*. Similarly, the occurrence of *examine casually* will disable *examine thoroughly*. In other words, there is a choice between these two activities. Transition *examine thoroughly* is executed for requests that are suspicious or complex. Straightforward requests only need a casual examination. Firing *check ticket* does not disable any other transition, i.e., it can occur concurrently with *examine thoroughly* or *examine casually*. Transition *decide* is only enabled if both input places contain a token. The ticket needs to be checked (token in place *c4*) and the casual or thorough examination of the request has been conducted (token in place *c3*). Hence, the process synchronizes before making a decision. Transition *decide* consumes two tokens and produces one token for *c5*. Three transitions share *c5* as an input place, thus modeling the three possible outcomes of the decision. The requested compensation is paid (transition *pay compensation* fires), the request is declined (transition *reject request* fires), or further processing is needed (transition *reinitiate request* fires). In the latter case the process returns to the state marking places *c1* and *c2*: transition *reinitiate request* consumes a token from *c5* and produces a token for each of its output places. This was the marking directly following the occurrence of *register request*. In principle, several iterations are possible. The process ends after paying the compensation or rejecting the request.

Process-Aware Information Systems
Process-Aware Information Systems (PAISs) include all software systems that support processes and not just isolated activities [49]. For example, ERP (Enterprise Resource Planning) systems (SAP, Oracle, etc.), BPM (Business Process Management) systems (Pegasystems, Bizagi, Appian, IBM BPM, etc.), WFM (Workflow Management) systems, CRM (Customer Relationship Management) systems, rule-based systems, call center software, high-end middle-

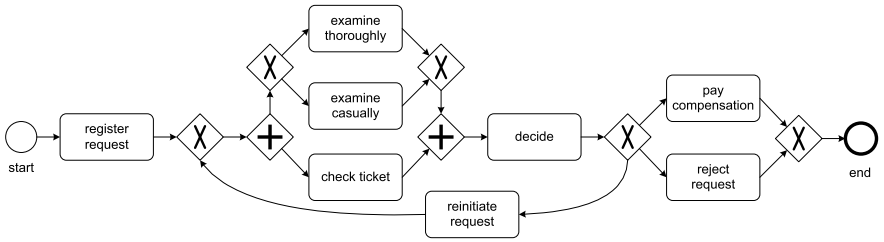


Fig. 2.3 The same process modeled in terms of BPMN

ware (WebSphere), etc. can be seen as process-aware. What these systems have in common is that there is a process notion present in the software (e.g., the completion of one activity triggers another activity) and that the information system is aware of the processes it supports (e.g., collecting information about flow times). This is very different from a database system, e-mail program, text editor, spreadsheet program, or currency transfer application. The latter set of example tools may be used to execute steps in some business process. However, these tools are not “aware” of the processes they are used in. Therefore, they cannot be actively involved in the management and orchestration of the processes they are used for.

A particular class of PAISs is formed by generic systems that are *driven by explicit process models*. Examples are BPM and WFM systems. WFM primarily focuses on the automation of business processes [76, 92, 151], whereas BPM has a broader scope: from process automation and process analysis to process management and the organization of work [50, 143, 187]. However, both BPM and WFM systems start from process models in a Petri-net or BPMN-like language. These models can be executed for any number of process instances. Changing the model corresponds (in theory) to automatically changing the process. This way flexibility and adaptability are support.

Note that ERP systems are often hybrid. They provide a WFM subsystem, but also support processes driven by (configurable) code rather than process models. What all PAISs have in common is that they can be configured in some way (through an explicit process model, via predefined settings, or using customization).

Figure 2.2 models the process as a Petri net. There exist many different notations for process models. Figure 2.3 models the same process in terms of a so-called BPMN diagram [110, 187]. The *Business Process Modeling Notation* (BPMN) uses explicit *gateways* rather than places to model the control-flow logic. The diamonds with a “$\times$” sign denote XOR split/join gateways, whereas diamonds with a “$+$” sign denote AND split/join gateways. The diamond directly following activity *register request* is an XOR-join gateway. This gateway is used to be able to “jump back”

after making the decision to reinitiate the request. After this XOR-join gateway there is an AND-split gateway to model that the checking of the ticket can be done in parallel with the selected examination type (thorough or casual). The remainder of the BPMN diagram is self explanatory as the behavior is identical to the Petri net described before.

Figures 2.2 and 2.3 show only the *control-flow*, i.e., the ordering of activities for the process described earlier. This is a rather limited view on business processes. Therefore, most modeling languages offer notations for modeling other perspectives such as the organizational or resource perspective (“The decision needs to be made by a manager”), the data perspective (“After four iteration always a decision is made unless more than 1 million Euro is claimed”), and the time perspective (“After two weeks the problem is escalated”). Although there are important differences between the various process modeling languages, we do not elaborate one these in this book. Instead, we refer to the systematic comparisons in the context of the *Workflow Patterns Initiative* [155, 191]. This allows us to focus on the role that process models play in process science, rather than worrying about notation. Although process mining can be used in a variety of applications domains, we often assume a BPM context for clarity. However, the techniques in this book can be used for *all* types of (discrete) events (e.g., in healthcare logistics, luggage handling systems, software analysis, smart maintenance, website analytics, and customer journey analysis).

What are process models used for?

- *insight*: while making a model, the modeler is triggered to view the process from various angles;
- *discussion*: the stakeholders use models to structure discussions;
- *documentation*: processes are documented for instructing people or certification purposes (cf. ISO 9000 quality management);
- *verification*: process models are analyzed to find errors in systems or procedures (e.g., potential deadlocks);
- *performance analysis*: techniques like simulation can be used to understand the factors influencing response times, service levels, etc.;
- *animation*: models enable end users to “play out” different scenarios and thus provide feedback to the designer;
- *specification*: models can be used to describe a PAIS before it is implemented and can hence serve as a “contract” between the developer and the end user/management; and
- *configuration*: models can be used to configure a system.

Clearly, process models play an important role in larger organizations. When redesigning processes and introducing new information systems, process models are used for a variety of reasons. Typically, two types of models are used: (a) *informal models* and (b) *formal models* (also referred to as “executable” models). Informal models are used for discussion and documentation whereas formal models

are used for analysis or enactment (i.e., the actual execution of process). On the one end of the spectrum there are “PowerPoint diagrams” showing high-level processes whereas on the other end of the spectrum there are process models captured in executable code. Whereas informal models are typically ambiguous and vague, formal models tend to have a rather narrow focus or are too detailed to be understandable by the stakeholders. The lack of alignment between both types of models has been discussed extensively in BPM literature [49, 70, 132, 137, 154, 187, 193]. Here, we would like to provide another view on the matter. Independent of the kind of model—informal or formal—one can reflect on the alignment between model and reality. A process model used to configure a workflow management system is probably well-aligned with reality as the model is used to force people to work in a particular way. Unfortunately, most hand-made models are disconnected from reality and provide only an idealized view on the processes at hand. Moreover, also formal models that allow for rigorous analysis techniques may have little to do with the actual process.

The value of models is limited if too little attention is paid to the alignment of model and reality. Process models become “paper tigers” when the people involved cannot trust them. For example, it makes no sense to conduct simulation experiments while using a model that assumes an idealized version of the real process. It is likely that—based on such an idealized model—incorrect redesign decisions are made. It is also precarious to start a new implementation project guided by process models that hide reality. A system implemented on the basis of idealized models is likely to be disruptive and unacceptable for end users. A nice illustration is the limited quality of most *reference models*. Reference models are used in the context of large enterprise systems such as SAP [37] but also to document processes for particular branches, cf. the NVVB (Nederlandse Vereniging Voor Burgerzaken) models describing the core processes in Dutch municipalities. The idea is that “best practices” are shared among different organizations. Unfortunately, the quality of such models leaves much to be desired. For example, the SAP reference model has very little to do with the processes actually supported by SAP. In fact, more than 20 percent of the SAP models contain serious flaws (deadlocks, livelocks, etc.) [101]. Such models are not aligned with reality and, thus, have little value for end users.

Given (a) the interest in process models, (b) the abundance of event data, and (c) the limited quality of hand-made models, it seems worthwhile to relate event data to process models. This way the actual processes can be discovered and existing process models can be evaluated and enhanced. This is precisely what process mining aims to achieve.

2.2 Process Mining

To position process mining, we first describe the so-called *BPM life-cycle* using Fig. 2.4. The life-cycle describes the different phases of managing a particular business process. In the *design* phase, a process is designed. This model is transformed

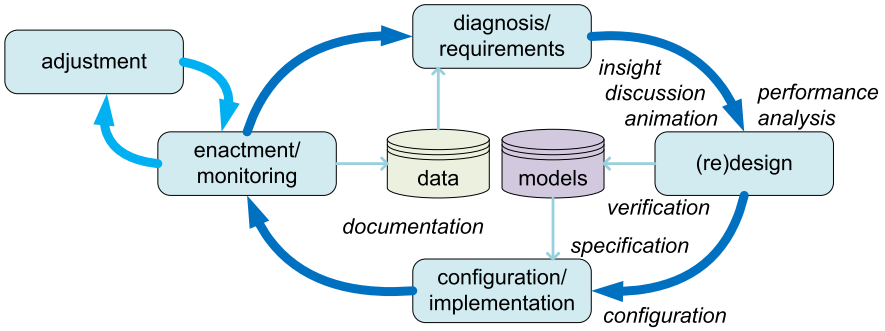


Fig. 2.4 The BPM life-cycle showing the different uses of process models

into a running system in the *configuration/implementation* phase. If the model is already in executable form and a WFM or BPM system is already running, this phase may be very short. However, if the model is informal and needs to be hard-coded in conventional software, this phase may take substantial time. After the system supports the designed processes, the *enactment/monitoring* phase starts. In this phase, the processes are running while being monitored by management to see if any changes are needed. Some of these changes are handled in the *adjustment* phase shown in Fig. 2.4. In this phase, the process is not redesigned and no new software is created; only predefined controls are used to adapt or reconfigure the process. The *diagnosis/requirements* phase evaluates the process and monitors emerging requirements due to changes in the environment of the process (e.g., changing policies, laws, competition). Poor performance (e.g., inability to meet service levels) or new demands imposed by the environment may trigger a new iteration of the BPM life-cycle starting with the *redesign* phase.

As Fig. 2.4 shows, process models play a dominant role in the (re)design and configuration/implementation phases, whereas data plays a dominant role in the enactment/monitoring and diagnosis/requirements phases. The figure also lists the different ways in which process models are used (as identified in Sect. 2.1). Until recently, there were few connections between the data produced while executing the process and the actual process design. In fact, in most organizations the diagnosis/requirements phase is not supported in a systematic and continuous manner. Only severe problems or major external changes will trigger another iteration of the life-cycle, and factual information about the current process is not actively used in redesign decisions. Process mining offers the possibility to truly “close” the BPM life-cycle. Data recorded by information systems can be used to provide a better view on the actual processes, i.e., deviations can be analyzed and the quality of models can be improved.

Process mining is a relative young research discipline that sits between machine learning and data mining on the one hand and process modeling and analysis on the other hand. The idea of process mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs readily available in today’s systems.

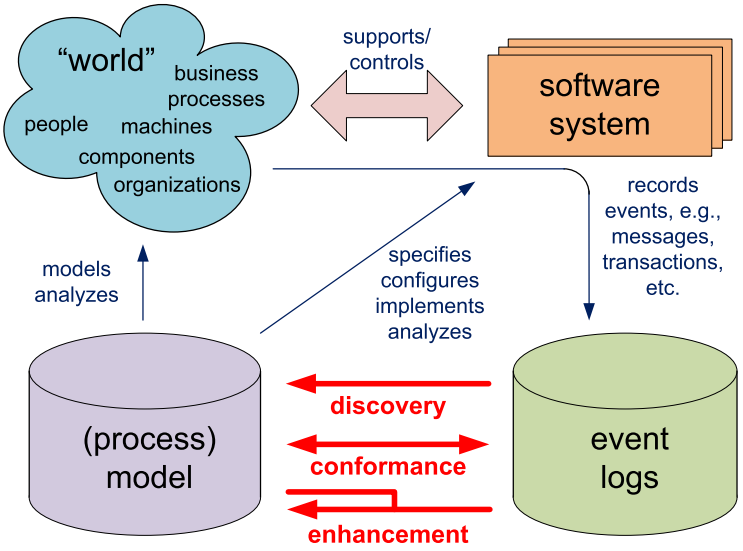


Fig. 2.5 Positioning of the three main types of process mining: *discovery*, *conformance*, and *enhancement*

Figure 2.5 shows that process mining establishes links between the actual processes and their data on the one hand and process models on the other hand. As explained in the previous chapter, the digital universe and the physical universe become more and more aligned. Today’s information systems log enormous amounts of events. Classical WFM systems (e.g., Staffware and COSA), BPM systems (e.g., BPM|one by Pallas Athena, SmartBPM by Pegasystems, FileNet, Global 360, and Teamwork by Lombardi Software), ERP systems (e.g., SAP Business Suite, Oracle E-Business Suite, and Microsoft Dynamics NAV), PDM systems (e.g., Windchill), CRM systems (e.g., Microsoft Dynamics CRM and SalesForce), middleware (e.g., IBM’s WebSphere and Cordys Business Operations Platform), and hospital information systems (e.g., Chipsoft and Siemens Soarian) provide detailed information about the activities that have been executed. Figure 2.5 refers to such data as *event logs*. All of the PAISs just mentioned directly provide such event logs. However, most information systems store such information in unstructured form, e.g., event data is scattered over many tables or needs to be tapped off from subsystems exchanging messages. In such cases, event data exist but some efforts are needed to extract them. Data extraction is an integral part of any process mining effort.

Let us assume that it is possible to *sequentially record events* such that each event refers to an *activity* (i.e., a well-defined step in the process) and is related to a particular *case* (i.e., a process instance). Consider, for example, the handling of requests for compensation modeled in Fig. 2.2. The cases are individual requests and per case a *trace* of events can be recorded. An example of a possible trace is $\langle$*register request*, *examine casually*, *check ticket*, *decide*, *reinitiate request*, *check ticket*, *examine thoroughly*, *decide*, *pay compensation*$\rangle$. Here activity names are used

to identify events. However, there are two *decide* events that occurred at different times (the fourth and eighth event of the trace), produced different results, and may have been conducted by different people. Obviously, it is important to distinguish these two decisions. Therefore, most event logs store additional information about events. In fact, whenever possible, process mining techniques use extra information such as the *resource* (i.e., person or device) executing or initiating the activity, the *timestamp* of the event, or *data elements* recorded with the event (e.g., the size of an order).

Event logs can be used to conduct three types of process mining as shown in Fig. 2.5.

The first type of process mining is *discovery*. A discovery technique takes an event log and produces a model without using any a-priori information. An example is the α-algorithm [157] that will be described in Chap. 6. This algorithm takes an event log and produces a Petri net explaining the behavior recorded in the log. For example, given sufficient example executions of the process shown in Fig. 2.2, the α-algorithm is able to automatically construct the Petri net without using any additional knowledge. If the event log contains information about resources, one can also discover resource-related models, e.g., a social network showing how people work together in an organization.

The second type of process mining is *conformance*. Here, an existing process model is compared with an event log of the same process. Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice versa. For instance, there may be a process model indicating that purchase orders of more than one million Euro require two checks. Analysis of the event log will show whether this rule is followed or not. Another example is the checking of the so-called “four-eyes” principle stating that particular activities should not be executed by one and the same person. By scanning the event log using a model specifying these requirements, one can discover potential cases of fraud. Hence, conformance checking may be used to detect, locate and explain deviations, and to measure the severity of these deviations. An example is the conformance checking algorithm described in [121]. Given the model shown in Fig. 2.2 and a corresponding event log, this algorithm can quantify and diagnose deviations.

The third type of process mining is *enhancement*. Here, the idea is to extend or improve an existing process model using information about the actual process recorded in some event log. Whereas conformance checking measures the alignment between model and reality, this third type of process mining aims at changing or extending the a-priori model. One type of enhancement is *repair*, i.e., modifying the model to better reflect reality. For example, if two activities are modeled sequentially but in reality can happen in any order, then the model may be corrected to reflect this. Another type of enhancement is *extension*, i.e., adding a new perspective to the process model by cross-correlating it with the log. An example is the extension of a process model with performance data. For instance, by using timestamps in the event log of the “request for compensation” process, one can extend Fig. 2.2 to show bottlenecks, service levels, throughput times, and frequencies. Similarly, Fig. 2.2 can be extended with information about resources, decision rules, quality metrics, etc.

As indicated earlier, process models such as depicted in Figs. 2.2 and 2.3 show only the control-flow. However, when extending process models, additional perspectives are added. Moreover, discovery and conformance techniques are not limited to control-flow. For example, one can discover a social network and check the validity of some organizational model using an event log. Hence, orthogonal to the three types of mining (discovery, conformance, and enhancement), different perspectives can be identified.

In the remainder, we consider the following *perspectives*.

- The *control-flow perspective* focuses on the control-flow, i.e., the ordering of activities. The goal of mining this perspective is to find a good characterization of all possible paths, e.g., expressed in terms of a Petri net or some other notation (e.g., EPCs, BPMN, and UML ADs).
- The *organizational perspective* focuses on information about resources hidden in the log, i.e., which actors (e.g., people, systems, roles, and departments) are involved and how are they related. The goal is to either structure the organization by classifying people in terms of roles and organizational units or to show the social network.
- The *case perspective* focuses on properties of cases. Obviously, a case can be characterized by its path in the process or by the originators working on it. However, cases can also be characterized by the values of the corresponding data elements. For example, if a case represents a replenishment order, it may be interesting to know the supplier or the number of products ordered.
- The *time perspective* is concerned with the timing and frequency of events. When events bear timestamps it is possible to discover bottlenecks, measure service levels, monitor the utilization of resources, and predict the remaining processing time of running cases.

Note that the different perspectives are partially overlapping and non-exhaustive. Nevertheless, they provide a good characterization of the aspects that process mining aims to analyze.

In most examples given thus far it is assumed that process mining is done *off-line*, i.e., processes are analyzed afterward to see how they can be improved or better understood. However, more and more process mining techniques can also be used in an *online* setting. We refer to this as *operational support*. An example is the detection of non-conformance at the moment the deviation actually takes place. Another example is time prediction for running cases, i.e., given a partially executed case the remaining processing time is estimated based on historic information of similar cases. This illustrates that the “process mining spectrum” is broad and not limited to process discovery. In fact, today’s process mining techniques are indeed able to support the whole BPM life-cycle shown in Fig. 2.4. Process mining is not only relevant for the design and diagnosis/requirements phases, but also for the enactment/monitoring and adjustment phases.

2.3 Analyzing an Example Log

After providing an overview of process mining and positioning it in the broader BPM discipline, we use the event log shown in Table 2.1 to clarify some of the foundational concepts. The table shows just a fragment of a possible log corresponding to the handling of requests for compensation. Each line presents one event. Note that events are already grouped per case. Case 1 has five associated events. The first event of Case 1 is the execution of activity *register request* by Pete on December 30th 2010. Table 2.1 also shows a unique id for this event: 35654423. This is merely used for the identification of the event, e.g., to distinguish it from event 35654483 that also corresponds to the execution of activity *register request* (first event of second case). Table 2.1 shows a date and a timestamp for each event. In some event logs this information is more coarse-grained and only a date or partial ordering of events is given. In other logs there may be more elaborate timing information also showing when the activity was started, when it was completed, and sometimes even when it was offered to the resource. The times shown in Table 2.1 should be interpreted as completion times. In this particular event log, activities are considered to be atomic and the table does not reveal the duration of activities. In the table, each event is associated to a resource. In some event logs this information will be missing. In other logs more detailed information about resources may be stored, e.g., the role a resource has or elaborate authorization data. The table also shows the costs associated to events. This is an example of a data attribute. There may be many other data attributes. For example, in this particular example it would be interesting to record the outcome of the different types of examinations and checks. Another data element that could be useful for analysis is the amount of compensation requested. This could be an attribute of the whole case or stored as an attribute of the *register request* event.

Table 2.1 illustrates the typical information present in an event log. Depending on the process mining technique used and the questions at hand, only part of this information is used. The minimal requirements for process mining are that any event can be related to both a case and an activity and that events within a case are ordered. Hence, the “case id” and “activity” columns in Table 2.1 represent the bare minimum for process mining. By projecting the information in these two columns we obtain the more compact representation shown in Table 2.2. In this table, each case is represented by a sequence of activities also referred to as *trace*. For clarity the activity names have been transformed into single-letter labels, e.g., a denotes activity *register request*.

Process mining algorithms for process discovery can transform the information shown in Table 2.2 into process models. For instance, the basic α-algorithm [157] discovers the Petri net described earlier when providing it with the input data in Table 2.2. Figure 2.6 shows the resulting model with the compact labels just introduced. It is easy to check that all six traces in Table 2.2 are possible in the model. Let us replay the trace of the first case—$\langle a, b, d, e, h\rangle$—to show that the trace “fits” (i.e., conforms to) the model. In the initial marking shown in Fig. 2.6, a is indeed enabled because of the token in *start*. After firing a places *c1* and *c2* are marked,

Table 2.1 A fragment of some event log: each line corresponds to an event

Case id	Event id	Properties				
		Timestamp	Activity	Resource	Cost	...
1	35654423	30-12-2010:11.02	register request	Pete	50	...
	35654424	31-12-2010:10.06	examine thoroughly	Sue	400	...
	35654425	05-01-2011:15.12	check ticket	Mike	100	...
	35654426	06-01-2011:11.18	decide	Sara	200	...
	35654427	07-01-2011:14.24	reject request	Pete	200	...
2	35654483	30-12-2010:11.32	register request	Mike	50	...
	35654485	30-12-2010:12.12	check ticket	Mike	100	...
	35654487	30-12-2010:14.16	examine casually	Pete	400	...
	35654488	05-01-2011:11.22	decide	Sara	200	...
	35654489	08-01-2011:12.05	pay compensation	Ellen	200	...
3	35654521	30-12-2010:14.32	register request	Pete	50	...
	35654522	30-12-2010:15.06	examine casually	Mike	400	...
	35654524	30-12-2010:16.34	check ticket	Ellen	100	...
	35654525	06-01-2011:09.18	decide	Sara	200	...
	35654526	06-01-2011:12.18	reinitiate request	Sara	200	...
	35654527	06-01-2011:13.06	examine thoroughly	Sean	400	...
	35654530	08-01-2011:11.43	check ticket	Pete	100	...
	35654531	09-01-2011:09.55	decide	Sara	200	...
	35654533	15-01-2011:10.45	pay compensation	Ellen	200	...
4	35654641	06-01-2011:15.02	register request	Pete	50	...
	35654643	07-01-2011:12.06	check ticket	Mike	100	...
	35654644	08-01-2011:14.43	examine thoroughly	Sean	400	...
	35654645	09-01-2011:12.02	decide	Sara	200	...
	35654647	12-01-2011:15.44	reject request	Ellen	200	...
5	35654711	06-01-2011:09.02	register request	Ellen	50	...
	35654712	07-01-2011:10.16	examine casually	Mike	400	...
	35654714	08-01-2011:11.22	check ticket	Pete	100	...
	35654715	10-01-2011:13.28	decide	Sara	200	...
	35654716	11-01-2011:16.18	reinitiate request	Sara	200	...
	35654718	14-01-2011:14.33	check ticket	Ellen	100	...
	35654719	16-01-2011:15.50	examine casually	Mike	400	...
	35654720	19-01-2011:11.18	decide	Sara	200	...
	35654721	20-01-2011:12.48	reinitiate request	Sara	200	...
	35654722	21-01-2011:09.06	examine casually	Sue	400	...
	35654724	21-01-2011:11.34	check ticket	Pete	100	...
	35654725	23-01-2011:13.12	decide	Sara	200	...
	35654726	24-01-2011:14.56	reject request	Mike	200	...

Table 2.1 (Continued)

Case id	Event id	Properties				
		Timestamp	Activity	Resource	Cost	...
6	35654871	06-01-2011:15.02	register request	Mike	50	...
	35654873	06-01-2011:16.06	examine casually	Ellen	400	...
	35654874	07-01-2011:16.22	check ticket	Mike	100	...
	35654875	07-01-2011:16.52	decide	Sara	200	...
	35654877	16-01-2011:11.47	pay compensation	Mike	200	...
...	...	...	...	...	...	...

Table 2.2 A more compact representation of log shown in Table 2.1: $a =$ *register request*, $b =$ *examine thoroughly*, $c =$ *examine casually*, $d =$ *check ticket*, $e =$ *decide*, $f =$ *reinitiate request*, $g =$ *pay compensation*, and $h =$ *reject request*

Case id	Trace
1	$\langle a, b, d, e, h\rangle$
2	$\langle a, d, c, e, g\rangle$
3	$\langle a, c, d, e, f, b, d, e, g\rangle$
4	$\langle a, d, b, e, h\rangle$
5	$\langle a, c, d, e, f, d, c, e, f, c, d, e, h\rangle$
6	$\langle a, c, d, e, g\rangle$
...	...

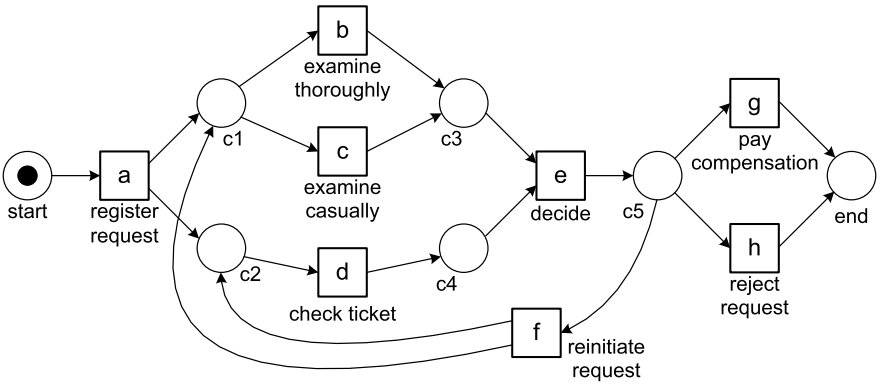


Fig. 2.6 The process model discovered by the α-algorithm [157] based on the set of traces $\{\langle a, b, d, e, h\rangle, \langle a, d, c, e, g\rangle, \langle a, c, d, e, f, b, d, e, g\rangle, \langle a, d, b, e, h\rangle, \langle a, c, d, e, f, d, c, e, f, c, d, e, h\rangle, \langle a, c, d, e, g\rangle\}$

i.e., both places contain a token. b is enabled at this marking and its execution results in the marking with tokens in *c2* and *c3*. Now we have executed $\langle a, b\rangle$ and the sequence $\langle d, e, h\rangle$ remains. The next event d is indeed enabled and its execution results in the marking enabling e (tokens in places *c3* and *c4*). Firing e results in the marking with one token in *c5*. This marking enables the final event h in the trace. After executing h, the case ends in the desired final marking with just a token in

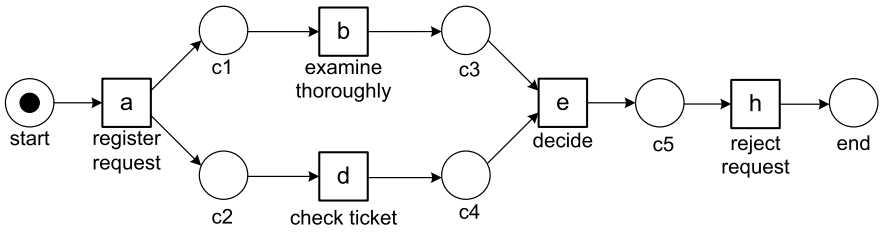


Fig. 2.7 The process model discovered by the α-algorithm based on cases 1 and 4, i.e., the set of traces $\{\langle a, b, d, e, h\rangle, \langle a, d, b, e, h\rangle\}$

place *end*. Similarly, it can be checked that the other five traces shown in Table 2.2 are also possible in the model and that all of these traces result in the marking with just a token in place *end*.

The Petri net shown in Fig. 2.6 also allows for traces not present in Table 2.2. For example, the traces $\langle a, d, c, e, f, b, d, e, g\rangle$ and $\langle a, c, d, e, f, c, d, e, f, c, d, e, f, c, d, e, f, b, d, e, g\rangle$ are also possible. This is a desired phenomenon as the goal is *not* to represent just the *particular set of example traces* in the event log. Process mining algorithms need to generalize the behavior contained in the log to show the most likely underlying model that is not invalidated by the next set of observations. One of the challenges of process mining is to balance between “overfitting” (the model is too specific and only allows for the “accidental behavior” observed) and “underfitting” (the model is too general and allows for behavior unrelated to the behavior observed).

When comparing the event log and the model, there seems to be a good balance between “overfitting” and “underfitting”. All cases start with a and end with either g or h. Every e is preceded by d and one of the examination activities (b or c). Moreover, e is followed by f, g, or h. The repeated execution of b or c, d, and e suggests the presence of a loop. These characteristics are adequately captured by the net of Fig. 2.6.

Let us now consider an event log consisting of only two traces $\langle a, b, d, e, h\rangle$ and $\langle a, d, b, e, h\rangle$, i.e., cases 1 and 4 of the original log. For this log, the α-algorithm constructs the Petri net shown in Fig. 2.7. This model only allows for two traces and these are exactly the ones in the small event log. b and d are modeled as being concurrent because they can be executed in any order. For larger and more complex models it is important to discover concurrency. Not modeling concurrency typically results in large “Spaghetti-like” models in which the same activity needs to be duplicated.[1]

The α-algorithm is just one of many possible process discovery algorithms. For real-life logs more advanced algorithms are needed to better balance between “overfitting” and “underfitting” and to deal with “incompleteness” (i.e., logs containing

[1] See, for example, Figs. 14.1 and 14.10 to understand why we use the term “Spaghetti” to refer to models that are difficult to comprehend.

Table 2.3 Another event log: cases 7, 8, and 10 are not possible according to Fig. 2.6

Case id	Trace
1	$\langle a, b, d, e, h\rangle$
2	$\langle a, d, c, e, g\rangle$
3	$\langle a, c, d, e, f, b, d, e, g\rangle$
4	$\langle a, d, b, e, h\rangle$
5	$\langle a, c, d, e, f, d, c, e, f, c, d, e, h\rangle$
6	$\langle a, c, d, e, g\rangle$
7	$\langle \mathbf{a}, \mathbf{b}, \mathbf{e}, \mathbf{g}\rangle$
8	$\langle \mathbf{a}, \mathbf{b}, \mathbf{d}, \mathbf{e}\rangle$
9	$\langle a, d, c, e, f, d, c, e, f, b, d, e, h\rangle$
10	$\langle \mathbf{a}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{b}, \mathbf{d}, \mathbf{g}\rangle$

only a small fraction of the possible behavior due to the large number of alternatives) and "noise" (i.e., logs containing exceptional/infrequent behavior that should not automatically be incorporated in the model). This book will describe several of such algorithms and guide the reader in selecting one. In this section, we used Petri nets to represent the discovered process models, because Petri nets are a succinct way of representing processes and have unambiguous and simple semantics. However, most mining techniques are independent of the desired representation. For instance, the discovered Petri net model shown in Fig. 2.6 can be (automatically) transformed into the BPMN model shown in Fig. 2.3.

As explained in Sect. 2.2, process mining is not limited to process discovery. Event logs can be used to check conformance and enhance existing models. Moreover, different perspectives may be taken into account. To illustrate this, let us first consider the event log shown in Table 2.3. The first six cases are as before. It is easy to see that Case 7 with trace $\langle a, b, e, g\rangle$ is not possible according to the model in Fig. 2.6. The model requires the execution of d before e, but d did not occur. This means that the ticket was not checked at all before making a decision and paying compensation. Conformance checking techniques aim at discovering such discrepancies [121]. When checking the conformance of the remainder of the event log it can also be noted that cases 8 and 10 do not conform either. Case 9 conforms although it is not identical to one of the earlier traces. Trace $\langle a, b, d, e\rangle$ (i.e., Case 8) has the problem that no concluding action was taken (rejection or payment). Trace $\langle a, c, d, e, f, b, d, g\rangle$ (Case 10) has the problem that the airline paid compensation without making a final decision. Note that conformance can be viewed from two angles: (a) the model does not capture the real behavior ("the model is wrong") and (b) reality deviates from the desired model ("the event log is wrong"). The first viewpoint is taken when the model is supposed to be *descriptive*, i.e., capture or predict reality. The second viewpoint is taken when the model is *normative*, i.e., used to influence or control reality.

The original event log shown in Table 2.1 also contains information about resources, timestamps and costs. Such information can be used to discover other perspectives, check the conformance of models that are not pure control-flow models, and to extend models with additional information. For example, one could derive

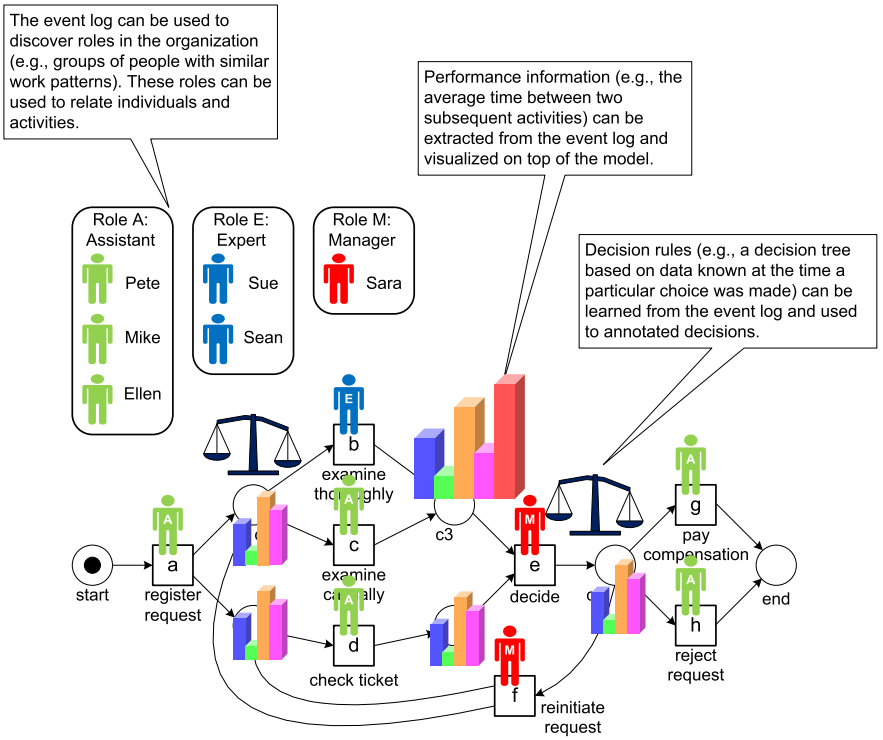


Fig. 2.8 The process model extended with additional perspectives: the organizational perspective (“What are the organizational roles and which resources are performing particular activities?”), the case perspective (“Which characteristics of a case influence a particular decision?”), and the time perspective (“Where are the bottlenecks in my process?”)

a social network based on the interaction patterns between individuals. The social network can be based on the “handover of work” metric, i.e., the more frequent individual x performed an activity that is causally followed by an activity performed by individual y, the stronger the relation between x and y is [159].

Figure 2.8 illustrates the way in which a control-flow oriented model can be extended with the three other main perspectives mentioned in Sect. 2.2. Analysis of the event log shown in Table 2.1 may reveal that Sara is the only one performing the activities *decide* and *reinitiate* request. This suggests that there is a “manager role” and that Sara is the only one having this role. Activity *examine thoroughly* is performed only by Sue and Sean. This suggests some “expert role” associated to this activity. The remaining activities are performed by Pete, Mike and Ellen. This suggests some “assistant role” as shown in Fig. 2.8. Techniques for organizational process mining [130] will discover such organizational structures and relate activities to resources through roles. By exploiting resource information in the log, the organizational perspective can be added to the process model. Similarly, information on timestamps and frequencies can be used to add performance related information

to the model. Figure 2.8 sketches that it is possible to measure the time that passes between an examination (activities *b* or *c*) and the actual decision (activity *e*). If this time is remarkably long, process mining can be used to identify the problem and discover possible causes. If the event log contains case-related information, this can be used to further analyze the decision points in the process. For instance, through decision point analysis it may be learned that requests for compensation of more than € 800 tend to be rejected.

Using process mining, the different perspectives can be cross-correlated to find surprising insights. Examples of such findings could be: “requests examined by Sean tend to be rejected more frequently”, “requests for which the ticket is checked after examination tend to take much longer”, “requests of less than € 500 tend to be completed without any additional iterations”. Moreover, these perspectives can also be linked to conformance questions. For example, it may be shown that Pete is involved in relatively many incorrectly handled requests. These examples show that privacy issues need to be considered when analyzing event logs with information about individuals (see Sect. 9.3.3).

2.4 Play-In, Play-Out, and Replay

One of the key elements of process mining is the emphasis on establishing a strong relation between a process model and “reality” captured in the form of an event log. Inspired by the terminology used by David Harel in the context of Live Sequence Charts [70], we use the terms *Play-In*, *Play-Out*, and *Replay* to reflect on this relation. Figure 2.9 illustrates these three notions.

Play-Out refers to the classical use of process models. Given a Petri net, it is possible to generate behavior. The traces in Table 2.2 could have been obtained by repeatedly “playing the token game” using the Petri net of Figure 2.6. Play-Out can be used both for the analysis and the enactment of business processes. A workflow engine can be seen as a “Play-Out engine” that controls cases by only allowing the “moves” allowed according to the model. Hence, Play-Out can be used to enact operational processes using some executable model. Simulation tools also use a Play-Out engine to conduct experiments. The main idea of simulation is to repeatedly run a model and thus collect statistics and confidence intervals. Note that a simulation engine is similar to a workflow engine. The main difference is that the simulation engine interacts with a modeled environment whereas the workflow engine interacts with the real environment (workers, customers, etc.). Also classical verification approaches using exhaustive state-space analysis—often referred to as model checking [30]—can be seen as Play-Out methods.

Play-In is the opposite of Play-Out, i.e., example behavior is taken as input and the goal is to construct a model. Play-In is often referred to as *inference*. The α-algorithm and other process discovery approaches are examples of Play-In techniques. Note that the Petri net of Fig. 2.6 can be derived automatically given an event log like the one in Table 2.2. Most data mining techniques use Play-In, i.e.,

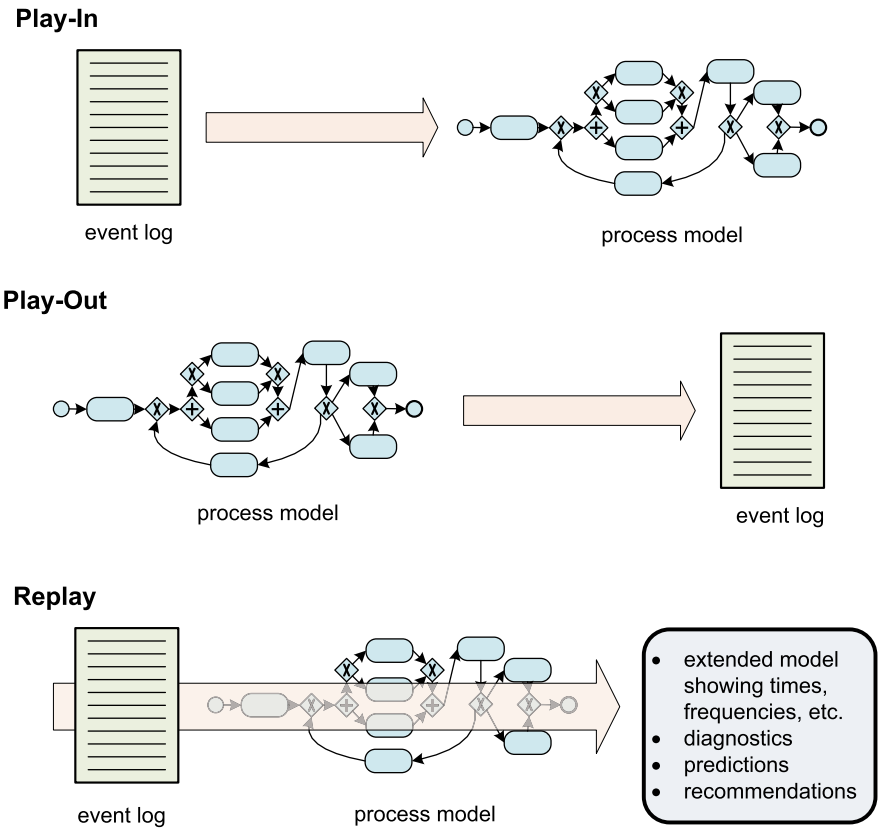


Fig. 2.9 Three ways of relating event logs (or other sources of information containing example behavior) and process models: *Play-In*, *Play-Out*, and *Replay*

a model is learned on the basis of examples. However, traditionally, data mining has not been concerned with process models. Typical examples of models are decision trees (“people that drink more than five glasses of alcohol and smoke more than 56 cigarettes tend to die young”) and association rules (“people that buy diapers also buy beer”). Unfortunately, it is not possible to use conventional data mining techniques to Play-In process models. Only recently, process mining techniques have become readily available to discover process models based on event logs.

Replay uses an event log *and* a process model as input. The event log is “replayed” on top of the process model. As shown earlier it is possible to replay trace $\langle a, b, d, e, h \rangle$ on the Petri net in Fig. 2.6; simply “play the token game” by forcing the transitions to fire (if possible) in the order indicated. An event log may be replayed for different purposes:

- *Conformance checking*: discrepancies between the log and the model can be detected and quantified by replaying the log. For instance, replaying trace

$\langle a, b, e, h \rangle$ on the Petri net in Fig. 2.6 will show that d should have happened but did not.

- *Extending the model with frequencies and temporal information.* By replaying the log one can see which parts of the model are visited frequently. Replay can also be used to detect bottlenecks. Consider, for example, the trace $\langle a^8, b^9, d^{20}, e^{21}, h^{21} \rangle$ in which the superscripts denote timestamps. By replaying the trace on top of Fig. 2.6 one can see that e was enabled at time 20 and occurred at time 21. The enabling of e was delayed by the time it took to complete d; although d was enabled already at time 8, it occurred only at time 20.
- *Constructing predictive models.* By replaying event logs one can build predictive models, i.e., for the different states of the model particular predictions can be made. For example, a predictive model learned by replaying many cases could show that the expected time until completion after enabling e is eight hours.
- *Operational support.* Replay is not limited to historic event data. One can also replay partial traces of cases still running. This can be used for detecting deviations at run-time, e.g., the partial trace $\langle a^8, e^{11} \rangle$ of a case that is still running will never fit into Fig. 2.6. Hence, an alert can be generated before the case completes. Similarly, it is possible to predict the remaining processing time or the likelihood of being rejected of a case having a partial trace, e.g., a partial executed case $\langle a^8, b^9 \rangle$ has an expected remaining processing time of 3.5 days and a 40 percent probability of being rejected. Such predictions can also be used to recommend suitable next steps to progress the case.

Desire lines in process models

A desire line—also known as the social trail—is a path that emerges through erosion caused by footsteps of humans (or animals). The width and amount of erosion of the path indicates how frequently the path is used. Typically, the desire line follows the shortest or most convenient path between two points. Moreover, as the path emerges more people are encouraged to use it, thus stimulating further erosion. Dwight Eisenhower is often mentioned as one of the persons using this emerging group behavior. Before becoming the 34th president of the United States, he was the president of Columbia University. When he was asked how the university should arrange the sidewalks to best interconnect the campus buildings, he suggested letting the grass grow between buildings and delay the creation of sidewalks. After some time the desire lines revealed themselves. The places where the grass was most worn by people’s footsteps were turned into sidewalks. In the same vein, replay can be used to show the *desire lines in processes*. The paths in the process model traveled most can be highlighted by using brighter colors or thicker arcs (cf. ProM’s Fuzzy Miner [66]).

An interesting question is how desire lines can be used to better manage business processes. Operational support, e.g., predictions and recommendations derived from historic information, can be used to reinforce successful behavior and thus create suitable “sidewalks” in processes.

2.5 Positioning Process Mining

The process mining spectrum is quite broad and extends far beyond process discovery and conformance checking. Process mining also connects data science and process science (see Fig. 1.7). As a result, it is inevitable that process mining objectives are overlapping with those of other approaches, methodologies, principles, methods, tools, and paradigms. For example, some will argue that “process mining is part of data mining”, but discussions on such inclusion relations are seldom useful and are often politically motivated. Most data mining tools do *not* provide process mining capabilities, most data mining books do *not* describe process mining techniques, and it seems that process mining techniques like conformance checking do *not* fit in any of the common definitions of data mining. It is comparable to claiming that “data mining is part of statistics”. Taking the transitive closure of both statements, we would even be able to conclude that process mining is part of statistics. Obviously, this does not make any sense. Making definitions all-encompassing does not help to provide actual analysis capabilities. Nevertheless, it is important to position process mining in the context of existing technologies and management approaches.

2.5.1 How Process Mining Compares to BPM

Business Process Management (BPM) is the discipline that combines approaches for the design, execution, control, measurement and optimization of business processes. Process mining can be best related to BPM by looking at the so-called BPM life-cycle in Fig. 2.4. Initially, the main focus of BPM was on process design and implementation [143]. Process modeling plays a key role in the (re)design phase and directly contributes to the configuration/implementation phase. Originally, BPM approaches had a tendency to be model-driven without considering the “evidence” hidden in the data.

There is now a clear trend in the BPM community to focus more on the enactment/monitoring, adjustment, and diagnosis/requirements phases. These phases are more data-driven and process mining techniques are frequently used in this part of the BPM life-cycle. Hence, process mining can easily be positioned in Fig. 2.4. However, process mining is *not* limited to BPM. Any process for which events can be recorded, is a candidate for process mining.

Learning more about Business Process Management (BPM)
Although process mining is not limited to BPM, both are clearly related. Hence, the interested reader may want to read more on BPM. Developments in BPM have resulted in a well-established set of principles, methods and tools that combine knowledge from information technology, management sciences and industrial engineering for the purpose of improving business processes. BPM can be viewed as a continuation of the Workflow Management (WFM) wave in the 1990s. The survey paper [143] structures the BPM field using 20 *BPM Use Cases* and describes the development of the field since the late 1970s. For more details, we refer to the following BPM/WFM books that served as milestones in the evolution of the field:

- *Workflow Management: Modeling Concepts, Architecture, and Implementation* [76]: first comprehensive WFM book focusing on the different workflow perspectives and the MOBILE language,
- *Production Workflow: Concepts and Techniques* [92]: book on production WFM systems closely related to IBM's workflow products,
- *Business Process Management: Models, Techniques, and Empirical Studies* [152]: edited book that served as the basis for the BPM conference series,
- *Workflow Management: Models, Methods, and Systems* [151]: most cited WFM book using a Petri net-based approach to model, analyze and enact workflow processes,
- *Workflow-based Process Controlling: Foundation, Design and Application of workflow-driven Process Information Systems* [192]: book relating WFM systems to operational performance,
- *Process-Aware Information Systems: Bridging People and Software through Process Technology* [49]: edited book on process-aware information systems,
- *Business Process Management: The Third Wave* [127]: visionary book linking management perspectives to the π-calculus,
- *Business Process Management: Concepts, Languages, Architectures* [187]: book presenting the foundations of BPM, including different languages and architectures,
- *Modern Business Process Automation: YAWL and its Support Environment* [132]: book based on YAWL and the workflow patterns,
- *Handbooks on Business Process Management* [180, 181]: edited handbooks covering the broader BPM discipline,
- *Process Management: A Guide for the Design of Business Processes* [18]: book on the design of process-oriented organizations,
- *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies* [115]: book on supporting flexibility in process-aware information systems, and
- *Fundamentals of Business Process Management* [50]: tutorial-style book covering the whole BPM life-cycle.

2.5.2 How Process Mining Compares to Data Mining

Data mining techniques aim to analyze (often large) data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner [69]. Like process mining, data mining is *data-driven*. However, unlike process mining, mainstream data mining techniques are typically *not process-centric*. Process models expressed in terms of Petri nets or BPMN diagrams cannot be discovered or analyzed in any way by the main data mining tools.

There are a few data mining techniques that come close to process mining. Examples are sequence and episode mining. However, these techniques do not consider end-to-end processes. Through process mining, it becomes easier to apply data mining techniques to event data. For example, decision rules can be learned using standard data mining tools after the control-flow backbone (e.g., a Petri net) has been learned using a process mining tool. *RapidProM*, available through the RapidMiner Marketplace, shows that process mining and data mining can be combined in various ways. Chapter 4 discusses the relation in more detail.

2.5.3 How Process Mining Compares to Lean Six Sigma

Lean Six Sigma is a methodology that combines ideas from *lean manufacturing* and *Six Sigma*. The idea is to improve performance by systematically removing waste. Lean principles originate from the Japanese manufacturing industry. The *Toyota Production System* (TPS) is a well-known example of a lean manufacturing approach developed by Taiichi Ohno and Eiji Toyoda between 1948 and 1975. The main objectives of the TPS are to eliminate “muri” (overburdening of people and equipment), “mura” (unevenness in operations), and “muda” (waste). The emphasis is on waste (“muda”) reduction. Typically, seven types of waste are mentioned in this context [109]:

- *Transportation waste*: Each time a product is moved, it encounters the risk of being damaged, lost, delayed, etc. Transportation does not make any transformation to the product that the consumer is willing to pay for (except for the final delivery).
- *Inventory waste*: Inventory may exist in the form of raw materials, work-in-progress, or finished goods. Inventory that is not being actively processed can be considered as waste because it consumes capital and space.
- *Motion waste*: Resources (equipment and people) that are used in the production processes suffer from “wear and tear”. Unnecessary activities (e.g., transformation and double work) result in additional degradation of resources and increase the risk of incidents (e.g., accidents).
- *Unnecessary waiting*: Whenever goods are not in transport or being processed, they are waiting. In traditional processes, a large part of an individual product’s life is spent waiting to be worked on. The total flow time of a case is often orders of magnitude larger than the sum of all service times.

- *Over-processing waste*: All additional efforts done for a product not directly required by the customer are considered as waste. This includes using components that are more precise, complex, of higher quality, and thus more expensive than absolutely required.
- *Overproduction waste*: Producing more than what is required by the customers at a particular time is a potential form of waste. Overproduction may lead to excess inventory and the customer's preferences may change over time making products outdated or less valuable.
- *Defects*: Rework, scrap, missing parts, poor work instructions, and correction activities are defects that can increase the costs of a product drastically.

Various additional types of waste have been identified. Although the terminology is oriented towards production processes and physical products, the same principles can be used for information/financial services, administrative work, and other BPM-like processes. The above examples illustrate that the focus of lean manufacturing is on *eliminating all non-value added activities*. Six Sigma focuses on *improving the quality of value added activities*. Both complement each other and are combined in Lean Six Sigma.

What does “Six Sigma” mean?
Today the term “Six Sigma” refers to a broad set of tools, techniques and methods to improve the quality of processes [113]. Six Sigma was originally developed by Motorola in the early 1980s and extended by many others. The term “Six Sigma” refers to the initial goal set by Motorola to minimize defects. In fact, the σ in “Six Sigma” refers to the standard deviation of a normal distribution. Given a normal distribution, 68.3% of the values lie within 1 standard deviation of the mean, i.e., a random draw from normal distribution with a mean value of μ and a standard deviation of σ has a probability of 0.683 to be in the interval $[\mu - \sigma, \mu + \sigma]$. Given the same normal distribution, 95.45% of randomly sampled values lie within two standard deviations of the mean, i.e., $[\mu - 2\sigma, \mu + 2\sigma]$, and 99.73% of the values lie within three standard deviations of the mean, i.e., $[\mu - 3\sigma, \mu + 3\sigma]$. The traditional quality paradigm in manufacturing defines a process as “capable” if the process's natural spread, plus and minus three σ, was less than the engineering tolerance. So, if deviations of up to three times the standard deviations are allowed, then on average 2700 out of one million cases will have a defect (i.e., samples outside the $[\mu - 3\sigma, \mu + 3\sigma]$ interval). Six Sigma aims to create processes were the standard deviation is so small that any value within 6 standard deviations of the mean can be considered as non-defective. In the literature, often a 1.5 sigma shift (to accommodate for long term variations and decreasing quality) is taken into account [113]. This results in the following table:

Quality level	Defective Parts per Million Opportunities (DPMO)	Percentage passed
One Sigma	690,000 DPMO	31%
Two Sigma	308,000 DPMO	69.2%
Three Sigma	66,800 DPMO	93.32%
Four Sigma	6,210 DPMO	99.379%
Five Sigma	230 DPMO	99.977%
Six Sigma	3.4 DPMO	99.9997%

A process that “runs at One Sigma” has less than 690,000 defective cases per million cases, i.e., at least 31% of the cases are handled properly. A process that “runs at Six Sigma” has only 3.4 defective cases per million cases, i.e., on average 99.9997% of the cases are handled properly.

A typical Lean Six Sigma project follows the so-called *DMAIC* approach consisting of five steps:

- *Define* the problem and set targets,
- *Measure* key performance indicators and collect data,
- *Analyze* the data to investigate and verify cause-and-effect relationships,
- *Improve* the current process based on this analysis, and
- *Control* the process to minimize deviations from the target.

Numerous organizations heavily invested in (Lean) Six Sigma training over the past decade. Based on Karate-like skill levels (green belt, black belt, etc.), certification programs were implemented. Unfortunately, the actual techniques are typically very basic (from a data science point of view). As a result, many consider Lean Six Sigma training as a management fad. Fortunately, process mining can be used as a tool to add more substance to the methodology. For example, process discovery can be used to eliminate all non-value added activities and reduce waste. If the relevant events are being recorded, we can visualize unnecessary waiting and rework. Conformance checking can also improve the quality of value added activities. Deviations can be found and diagnosed easily, provided that the event data and normative process models are present.

Related to Lean Six Sigma are management approaches such as: *Continuous Process Improvement* (CPI), *Total Quality Management* (TQM), *5S* (workplace organization method characterized by the terms Sort, Straighten, Shine, Standardize, and Sustain), *Kaizen* (another continuous improvement method), and *Theory of Constraints* (management paradigm by Eliyahu Goldratt based in the idea that “a chain is no stronger than its weakest link”, thus focusing on the constraints limiting performance). What these approaches have in common is that processes are “put under a microscope” to see whether further improvements are possible. Clearly, process mining can help to analyze deviations and inefficiencies.

2.5.4 How Process Mining Compares to BPR

Business Process Reengineering (BPR) is a management approach developed by people like Michael Hammer [68]. BPR is characterized by four key words: *fundamental*, *radical*, *dramatic* and *process* [151]. The keyword *fundamental* indicates that, when revitalizing a business process, it is of great importance always to ask the basic question: Why are we doing this, and why are we doing it in this way? *Radical* means that the reengineered process must represent a complete break from the current way of working. BPR does not advocate to gradually improve existing processes: It aims at finding by completely new ones. The third keyword also refers to the fact that BPR does not aim at marginal or superficial changes. Changes must be *dramatic* in terms of costs, service and quality. In order to achieve dramatic improvements, it is necessary to focus on the *processes* and not start from data or systems.

BPR is *process-centric*, i.e., the focus is on the process just like in process mining. However, BPR is *not data-driven*. It promotes “thinking outside the box” rather than analyzing data in great detail. Process mining helps to identify the problems and assists shareholders in defining improvement actions. However, process mining cannot come up with completely different ways of working (unless event data is enriched with domain knowledge).

2.5.5 How Process Mining Compares to Business Intelligence

Process mining can be positioned under the umbrella of *Business Intelligence* (BI). There is no clear definition for BI. On the one hand, it is a very broad term that includes anything that aims at providing actionable information that can be used to support decision making. On the other hand, vendors and consultants tend to conveniently skew the definition towards a particular tool or methodology. Process mining provides innovations highly relevant for the next generation of BI techniques. However, it is important to note that current BI tools are not really “intelligent” and do not provide any process mining capabilities. The focus is on querying and reporting combined with simple visualization techniques showing dashboards and scorecards. Some systems provide data mining capabilities or support *Online Analytical Processing* (OLAP). OLAP tools are used to view multidimensional data from different angles. On the one hand, it is possible to aggregate and consolidate data to create high-level reports. On the other hand, OLAP tools can drill down into the data to find detailed information. There are approaches that combine process mining with OLAP to create and analyze so-called *process cubes* filled with event data (see Sect. 12.4).

Under the BI umbrella, many fancy terms have been introduced to refer to rather simple reporting and dashboard tools. *Business Activity Monitoring* (BAM) refers to the real-time monitoring of business processes. *Corporate Performance Management* (CPM) is another buzzword for measuring the performance of a process or

organization. Typically, CPM focuses on financial aspects. Recently, more and more software vendors started to use the term “analytics” to refer to advanced BI capabilities. *Visual analytics* focuses on the analysis of large amounts of data while exploiting the remarkable capabilities of humans to visually identify patterns and trends. *Predictive analytics* uses historic data to make forecasts. Clearly, process mining also aims at providing advanced analytics and some process mining techniques also heavily rely on advanced visualization and human interpretation. Moreover, as will be demonstrated in Chapt. 10, process mining is not restricted to analyzing historic data and also includes operational support, i.e., providing predictions and recommendations in an online setting.

2.5.6 How Process Mining Compares to CEP

Process mining complements *Complex Event Processing* (CEP). CEP combines data from multiple sources to infer events or patterns that suggest higher-level events. The goal of CEP is to identify meaningful events (such as opportunities or threats) and respond to them as quickly as possible, e.g., immediately generate an alert when a combination of events occurs. CEP can be used as a preprocessing step for process mining, i.e., low level event data with many (seemingly) meaningless events can be converted into higher-level event streams used by process mining techniques (online or offline). CEP is particularly useful if there are many (low-level) events. By reducing torrents of event data to manageable streams or logs, analysis becomes easier.

2.5.7 How Process Mining Compares to GRC

Whereas management approaches such as Lean Six Sigma and BPR mainly aim at improving operational performance, e.g., reducing flow time and defects, organizations are also putting increased emphasis on *corporate governance*, *risk*, and *compliance*. The frequently used acronym *GRC* is composed of the pillars *Governance*, *Risk management* and *Compliance*, and refers to an organization’s capability to reliably achieve its objectives while addressing uncertainty and acting with integrity. *Governance* is the combination of culture, policies, processes, laws, and institutions that define the structure by which the organization is directed and managed. *Risk management* is the process of identifying, assessing, and prioritizing risks, as well as creating a plan for minimizing or eliminating the impact of negative events. *Compliance* is the act of adhering to, and demonstrating adherence to, external laws and regulations as well as corporate policies and procedures.

Major corporate and accounting scandals including those affecting Enron, Tyco, Adelphia, Peregrine, and WorldCom have fueled interest in more rigorous auditing practices. Legislation such as the *Sarbanes–Oxley Act* (SOX) of 2002 and the different *Basel Accords* was enacted in response to such scandals. The financial crisis

a few years ago also underscores the importance of verifying that organizations operate “within their boundaries”. Process mining techniques offer a means to a more rigorous compliance checking and ascertaining the validity and reliability of information about an organization’s core processes. Since conformance checking can be used to reveal deviations, defects, and near incidents, it is a valuable tool to check compliance and manage risks.

2.5.8 How Process Mining Compares to ABPD, BPI, WM, . . .

As described in the *Process Mining Manifesto* [75] by the IEEE Task Force on Process Mining, there are several alternative terms for process mining. Sometimes these terms are synonyms and at other times they refer to particular process mining tasks.

Automated Business Process Discovery (ABPD) is an example of such a term. ABPD was introduced by Gartner introduced in 2008 [84]. Often ABPD is used to refer to just process discovery (i.e., discovering process models from event data). This does not include bottleneck analysis, conformance checking, prediction, social network analysis, etc. Sometimes ABPD is used as a synonym for process mining. However, Fig. 2.5 clearly shows that process mining includes, for example, conformance checking. Moreover, later other forms of process mining will be described (e.g., prediction). Vendors that claim to support ABPD typically only uphold a fraction of the process mining spectrum covered in this book.

The term *Business Process Intelligence* (BPI) is used in different ways. It is often used as a synonym for process mining (perhaps with less emphasis on process models). See, for example, the annual International Workshop on Business Process Intelligence that has been running since 2005. Almost all papers presented at these BPI workshops use or propose process mining techniques.

BPI can also be understood as BI with a focus on analyzing operational processes. Some of the products positioned as BPI tools do not support discovery, i.e., performance data are mapped onto hand-made models. These tools assume a stable and known process model. Terms comparable to BPI are used by a range of vendors. For example, IBM’s Business Process Manager refers to the latter BPI-like functionality (i.e., without process discovery) as *Business Process Analytics* (BPA).

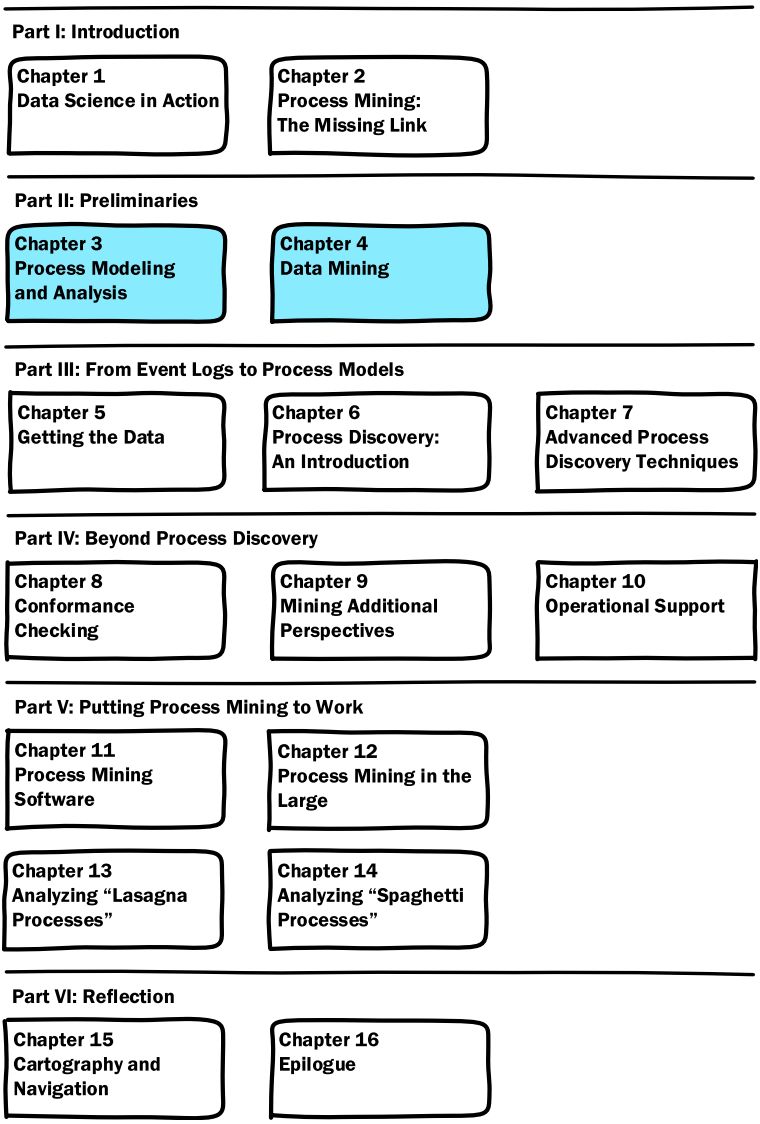
The term *Workflow Mining* (WM) was a precursor for process mining. It dates from a time where the main aim of process mining was the automatic configuration of a WFM system. Ideally, one would like to observe an existing process and automatically generate the corresponding executable workflow model. This view turned out to be too narrow and often unrealistic. Process mining has a much wider applicability, also in areas unrelated to WFM/BPM systems. Moreover, through discovery one can indeed find a skeleton of the workflow model. However, the model needs to be enriched with technical details to obtain an executable workflow. This makes the automated generation of workflow models less practical. Today, process mining is predominantly an approach for performance and conformance analysis (see Fig. 2.1). Therefore, the term WM is no longer actively used.

2.5.9 How Process Mining Compares to Big Data

In Chap. 1, we listed the “four V’s of Big Data”: Volume, Velocity, Variety, and Veracity (Fig. 1.4). These reflect the typical characteristics of some of the exciting new data sources interesting for analysis. Big Data does not focus on a particular type of analysis and is not limited to process-related data. However, Big Data infrastructures enable us to collect, store, and process huge event logs. Process mining tools can exploit such infrastructures to distribute large analysis tasks over multiple computing resources. For example, the MapReduce programming model can be used for discovery algorithms and the Hadoop Distributed File System (HDFS) can be used to store event data in a distributed fashion. In principle, one can use thousands of compute nodes to perform process mining analyses. Chapter 12 will elaborate on this.

The many acronyms in this section—BPM, BI, OLAP, TPS, BAM, CEP, CPM, CPI, TQM, SOX, etc.—are just a subset of the jargon used by business consultants and vendors. Some are just variations on the same theme, others emphasize a particular aspect. What can be distilled from the above is that there is a clear trend towards actually using the data available in today’s systems. The data is used to *reason about the process* and for *decision making within the process*. Moreover, the acronyms express a clear desire to get more *insight* into the actual processes, to *improve* them, and to make sure that they are *compliant*. Unfortunately, buzzwords are often used when the actual analysis capabilities are weak. When listening to a product presentation of conference talk, one is often tempted to play “buzzword bingo” (also known as bullshit bingo) illustrating that the foundational issues are not addressed. This book aims to provide a clear and refreshing view on the matter. Using recent breakthroughs in process mining, we will show that it is possible to simplify and unify the analysis of business processes based on facts. Moreover, the techniques and insights presented are directly applicable and are supported by process mining tools such as *ProM* (www.processmining.org).

Part II
Preliminaries



Process mining provides a bridge between data mining and process modeling and analysis. Therefore, we provide an introduction to both fields. Chapter 3 reviews various process modeling notations and their analysis. Chapter 4 explains the main data mining techniques.

Chapter 3
Process Modeling and Analysis

The plethora of process modeling notations available today illustrates the relevance of process modeling. Some organizations may use only informal process models to structure discussions and to document procedures. However, organizations that operate at a higher BPM maturity level use models that can be analyzed and used to enact operational processes. Today, most process models are made by hand and are not based on a rigorous analysis of existing process data. This chapter serves two purposes. On the one hand, preliminaries are presented that will be used in later chapters. For example, various process modeling notations are introduced and some analysis techniques are reviewed. On the other hand, the chapter reveals the limitations of classical approaches, thus motivating the need for process mining.

3.1 The Art of Modeling

In Sect. 1.3, we introduced the umbrella term “*process science*” to refer to the broader discipline that combines knowledge from information technology and knowledge from management sciences to improve and run operational processes. Many of the (sub)disciplines mentioned in Fig. 1.6 heavily rely on *modeling* using a variety of formalisms and notations. In this book, we will use transition systems, Petri nets, BPMN, C-nets, EPCs, YAWL, and process trees as example representations. Before providing a “crash course” in these process representations, we briefly reflect on the role of models and the limitations of modeling in process science.

Since the industrial revolution, productivity has been increasing because of technical innovations, improvements in the organization of work, and the use of information technology. Adam Smith (1723–1790) showed the advantages of the division of labor. Frederick Taylor (1856–1915) introduced the initial principles of scientific management. Henry Ford (1863–1947) introduced the production line for the mass production of “black T-Fords”. Around 1950 computers and digital communication infrastructures started to influence business processes. This resulted in dramatic changes in the organization of work and enabled new ways of doing business. Today, innovations in computing and communication are still the main drivers behind

W. van der Aalst, *Process Mining*, DOI 10.1007/978-3-662-49851-4_3

change in business processes. So, business processes have become more complex, heavily rely on information systems, and may span multiple organizations. Therefore, process modeling has become of the utmost importance. Process models assist in managing complexity by providing insight and documenting procedures. Information systems need to be configured and driven by precise instructions. Cross-organizational processes can only function properly if there is a common agreement on the required interactions. As a result, process models are widely used in today's organizations.

Operations management, and in particular *operation research*, is a branch of management science heavily relying on modeling. Here a variety of mathematical models ranging from linear programming and project planning to queueing models, Markov chains, and simulation are used. For example, the location of a warehouse is determined using linear programming, server capacity is added on the basis of queueing models, and an optimal route in a container terminal is determined using integer programming. Models are used to reason *about processes* (redesign) and to make decisions *inside processes* (planning and control). The models used in operations management are typically tailored towards a particular analysis technique and only used for answering a specific question. In contrast, process models in BPM typically serve *multiple* purposes. A process model expressed in BPMN may be used to discuss responsibilities, analyze compliance, predict performance using simulation, and configure a WFM system. However, BPM and operations management have in common that making a good model is “an art rather than a science”. Creating models is therefore a difficult and error-prone task. Typical errors include:

- *The model describes an idealized version of reality.* When modeling processes the designer tends to concentrate on the “normal” or “desirable” behavior. For example, the model may only cover 80% of the cases assuming that these are representative. Typically this is not the case as the other 20% may cause 80% of the problems. The reasons for such oversimplifications are manifold. The designer and management may not be aware of the many deviations that take place. Moreover, the perception of people may be biased, depending on their role in the organization. Hand-made models tend to be subjective, and often there is a tendency to make things too simple just for the sake of understandability.
- *Inability to adequately capture human behavior.* Although simple mathematical models may suffice to model machines or people working in an assembly line, they are inadequate when modeling people involved in multiple processes and exposed to multiple priorities [139, 163]. A worker who is involved in multiple processes needs to distribute his attention over multiple processes. This makes it difficult to model one process in isolation. Workers also do not work at constant speed. A well-known illustration of this is the so-called *Yerkes–Dodson law* that describes the relation between workload and performance of people [139]. In most processes one can easily observe that people will take more time to complete a task and effectively work fewer hours per day if there is hardly any work to do. Nevertheless, most simulation models sample service times from a fixed probability distribution and use fixed time windows for resource availability.

- *The model is at the wrong abstraction level.* Depending on the input data and the questions that need to be answered, a suitable abstraction level needs to be chosen. The model may be too abstract and thus unable to answer relevant questions. The model may also be too detailed, e.g., the required input cannot be obtained or the model becomes too complex to be fully understood. Consider, for example, a car manufacturer that has a warehouse containing thousands of spare parts. It may be tempting to model all of them in a simulation study to compare different inventory policies. However, if one is not aiming at making statements about a specific spare part, this is not wise. Typically it is very time consuming to change the abstraction level of an existing model. Unfortunately, questions may emerge at different levels of granularity.

These are just some of the problems organizations face when making models by hand. Only experienced designers and analysts can make models that have a good predictive value and can be used as a starting point for a (re)implementation or redesign. An inadequate model can lead to wrong conclusions. Therefore, we advocate the use of event data. Process mining allows for the extraction of models based on *facts*. Moreover, process mining does not aim at creating a single model of the process. Instead, it provides *various views on the same reality at different abstraction levels*. For example, users can decide to look at the most frequent behavior to get a simple model (“80% model”). However, they can also inspect the full behavior by deriving the “100% model” covering all cases observed. Similarly, abstraction levels can be varied to create different views. Process mining can also reveal that people in organizations do not function as “machines”. On the one hand, it may be shown that all kinds of inefficiencies take place. On the other hand, process mining can also visualize the remarkable flexibility of some workers to deal with problems and varying workloads.

3.2 Process Models

It is not easy to make good process models. Yet, they are important. Fortunately, process mining can facilitate the construction of better models in less time. Process discovery algorithms like the α-algorithm can automatically generate a process model. As indicated in Chap. 2, various process modeling notations exist. Sometimes the plethora of notations is referred to as the new “tower of Babel”. Therefore, we describe only some basic notations. This section does not aim to provide a complete overview of existing process modeling notations. We just introduce the notations that we will use in the remainder. We would like to stress that it is relatively easy to automatically translate process mining results into the desired notation. For example, although the α-algorithm produces a Petri net, it is easy to convert the result into a BPMN model, BPEL model, or UML Activity Diagram. Again we refer to the systematic comparisons in the context of the Workflow Patterns Initiative [155, 191] for details.

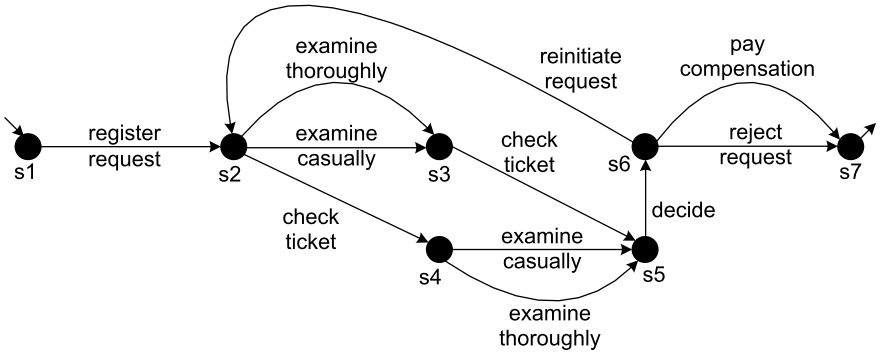


Fig. 3.1 A transition system having one initial state and one final state

In this section we focus on the control-flow perspective of processes. We assume that there is a set of *activity labels* $\mathscr{A}$. The goal of a process model is to decide *which activities* need to be executed and in *what order*. Activities can be executed sequentially, activities can be optional or concurrent, and the repeated execution of the same activity may be possible.

3.2.1 Transition Systems

The most basic process modeling notation is a *transition system*. A transition system consists of *states* and *transitions*. Figure 3.1 shows a transition system consisting of seven states. It models the handling of a request for compensation within an airline as described in Sect. 2.1. The states are represented by black circles. There is one initial state labeled *s1* and one final state labeled *s7*. Each state has a unique label. This label is merely an identifier and has no meaning. Transitions are represented by arcs. Each transition connects two states and is labeled with the name of an activity. Multiple arcs can bear the same label. For example, *check ticket* appears twice.

Definition 3.1 (Transition system) A *transition system* is a triplet $TS = (S, A, T)$ where S is the set of *states*, $A \subseteq \mathscr{A}$ is the set of *activities* (often referred to as *actions*), and $T \subseteq S \times A \times S$ is the set of *transitions*. $S^{start} \subseteq S$ is the set of *initial states* (sometimes referred to as “start” states), and $S^{end} \subseteq S$ is the set of *final states* (sometimes referred to as “accept” states).

The sets S^{start} and S^{end} are defined implicitly. In principle, S can be infinite. However, for most practical applications the state space is finite. In this case the transition system is also referred to as a Finite-State Machine (FSM) or a finite-state automaton.

The transition system depicted in Fig. 3.1 can be formalized as follows: $S = \{s1, s2, s3, s4, s5, s6, s7\}$, $S^{start} = \{s1\}$, $S^{end} = \{s7\}$, $A = \{$*register request*, *examine thoroughly*, *examine casually*, *check ticket*, *decide*, *reinitiate request*, *reject*

request, *pay compensation*}, and $T = \{(s1, \textit{register request}, s2), (s2, \textit{examine casually}, s3), (s2, \textit{examine thoroughly}, s3), (s2, \textit{check ticket}, s4), (s3, \textit{check ticket}, s5), (s4, \textit{examine casually}, s5), (s4, \textit{examine thoroughly}, s5), (s5, \textit{decide}, s6), (s6, \textit{reinitiate request}, s2), (s6, \textit{pay compensation}, s7), (s6, \textit{reject request}, s7)\}$.

Given a transition system one can reason about its behavior. The transition starts in one of the initial states. Any path in the graph starting in such a state corresponds to a possible *execution sequence*. For example, the path *register request*, *examine casually*, *check ticket* in Fig. 3.1 is an example of an execution sequence starting in state $s1$ and ending in $s5$. There are infinitely many execution sequences for this transition system. A path *terminates successfully* if it ends in one of the final states. A path *deadlocks* if it reaches a non-final state without any outgoing transitions. Note that the absence of deadlocks does not guarantee successful termination. The transition system may *livelock*, i.e., some transitions are still enabled but it is impossible to reach one of the final states.

Any process model with executable semantics can be mapped onto a transition system. Therefore, many notions defined for transition systems can easily be translated to higher-level languages such as Petri nets, BPMN, and UML activity diagrams. Consider, for example, the seemingly simple question: "When are two processes the same from a behavioral point of view". As shown in [176], many equivalence notions can be defined. *Trace equivalence* considers two transition systems to be equivalent if their execution sequences are the same. More refined notions like *branching bisimilarity* also take the moment of choice into account. These notions defined for transition systems can be used for any pair of process models as long as the models are expressed in a language with executable semantics (see also Sect. 6.3).

Transition systems are simple but have problems expressing concurrency succinctly. Suppose that there are n parallel activities, i.e., all n activities need to be executed but any order is allowed. There are $n!$ possible execution sequences. The transition system requires 2^n states and $n \times 2^{n-1}$ transitions. This is an example of the well-known "state explosion" problem [135]. Consider for example 10 parallel activities. The number of possible execution sequences is $10! = 3{,}628{,}800$, the number of reachable states is $2^{10} = 1024$, and the number of transitions is $10 \times 2^{10-1} = 5120$. The corresponding Petri net is much more compact and needs only 10 transitions and 10 places to model the 10 parallel activities. Given the concurrent nature of business processes, more expressive models like Petri nets are needed to adequately represent process mining results.

3.2.2 Petri Nets

Petri nets are the oldest and best investigated process modeling language allowing for the modeling of concurrency. Although the graphical notation is intuitive and simple, Petri nets are executable and many analysis techniques can be used to analyze them [82, 117, 149]. In the introduction we already showed an example Petri

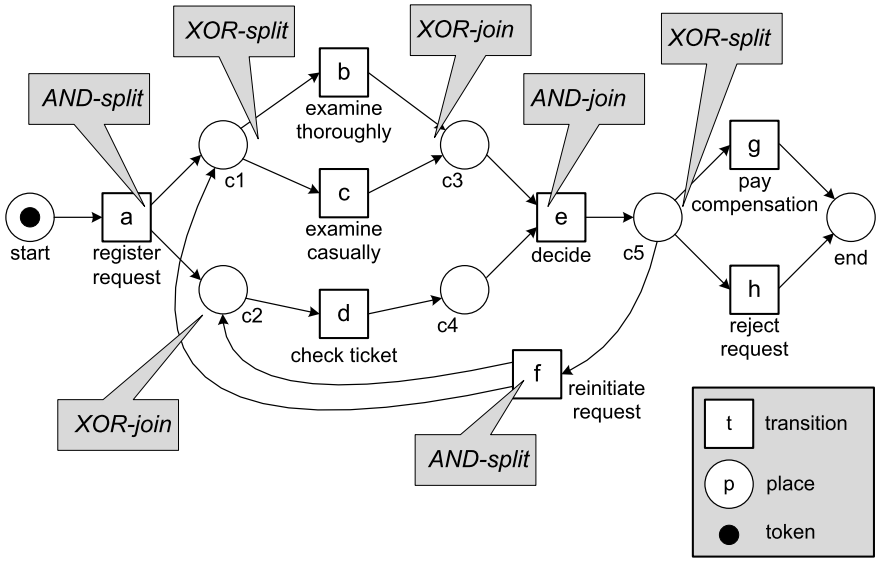


Fig. 3.2 A marked Petri net

net. Figure 3.2 shows the Petri net again with the various constructs highlighted. A Petri net is a bipartite graph consisting of *places* and *transitions*. The network structure is static, but, governed by the firing rule, *tokens* can flow through the network. The state of a Petri net is determined by the distribution of tokens over places and is referred to as its *marking*. In the initial marking shown in Fig. 3.2, there is only one token; *start* is the only marked place.

Definition 3.2 (Petri net) A *Petri net* is a triplet $N = (P, T, F)$ where P is a finite set of *places*, T is a finite set of *transitions* such that $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the *flow relation*. A *marked* Petri net is a pair (N, M), where $N = (P, T, F)$ is a Petri net and where $M \in \mathbb{B}(P)$ is a *multi-set* over P denoting the *marking* of the net. The set of all marked Petri nets is denoted $\mathcal{N}$.

The Petri net shown Fig. 3.2 can be formalized as follows: $P = \{start, c1, c2, c3, c4, c5, end\}$, $T = \{a, b, c, d, e, f, g, h\}$, and $F = \{(start, a), (a, c1), (a, c2), (c1, b), (c1, c), (c2, d), (b, c3), (c, c3), (d, c4), (c3, e), (c4, e), (e, c5), (c5, f), (f, c1), (f, c2), (c5, g), (c5, h), (g, end), (h, end)\}$.

Multi-sets
A marking corresponds to a multi-set of tokens. However, multi-sets are not only used to represent markings; later we will use multi-sets to model event logs where the same trace may appear multiple times. Therefore, we provide some basic notations used in the remainder.

A multi-set (also referred to as *bag*) is like a set in which each element may occur multiple times. For example, $[a, b^2, c^3, d^2, e]$ is the multi-set with nine elements: one a, two b's, three c's, two d's, and one e. The following three multi-set are identical: $[a, b, b, c^3, d, d, e]$, $[e, d^2, c^3, b^2, a]$, and $[a, b^2, c^3, d^2, e]$. Only the number of occurrences of each value matters, not the order. Formally, $\mathbb{B}(D) = D \to \mathbb{N}$ is the set of multi-sets (bags) over a finite domain D, i.e., $X \in \mathbb{B}(D)$ is a multi-set, where for each $d \in D$, $X(d)$ denotes the number of times d is included in the multi-set. For example, if $X = [a, b^2, c^3]$, then $X(b) = 2$ and $X(e) = 0$.

The sum of two multi-sets ($X \uplus Y$), the difference ($X \setminus Y$), the presence of an element in a multi-set ($x \in X$), and the notion of subset ($X \leq Y$) are defined in a straightforward way. For example, $[a, b^2, c^3, d] \uplus [c^3, d, e^2, f^3] = [a, b^2, c^6, d^2, e^2, f^3]$ and $[a, b] \leq [a, b^3, c]$. Moreover, we can also apply these operators to sets, where we assume that a set is a multi-set in which every element occurs exactly once. For example, $[a, b^2] \uplus \{b, c\} = [a, b^3, c]$.

The operators are also robust with respect to the domains of the multi-sets, i.e., even if X and Y are defined on different domains, $X \uplus Y$, $X \setminus Y$, and $X \leq Y$ are defined properly by extending the domain whenever needed.

The marking shown in Fig. 3.2 is $[start]$, i.e., a multi-set containing only one token. The dynamic behavior of such a marked Petri net is defined by the so-called *firing rule*. A transition is *enabled* if each of its input places contains a token. An enabled transition can *fire* thereby consuming one token from each input place and producing one token for each output place. Hence, transition a is enabled at marking $[start]$. Firing a results in the marking $[c1, c2]$. Note that one token is consumed and two tokens are produced. At marking $[c1, c2]$, transition a is no longer enabled. However, transitions b, c, and d have become enabled. From marking $[c1, c2]$, firing b results in marking $[c2, c3]$. Here, d is still enabled, but b and c not anymore. Because of the loop construct involving f there are infinitely many firing sequences starting in $[start]$ and ending in $[end]$.

Assume now that the initial marking is $[start^5]$. Firing a now results in the marking $[start^4, c1, c2]$. At this marking a is still enabled. Firing a again results in marking $[start^3, c1^2, c2^2]$. Transition a can fire five times in a row resulting in marking $[c1^5, c2^5]$. Note that after the first occurrence of a, also b, c, and d are enabled and can fire concurrently.

To formalize the firing rule, we introduce a notation for input (output) places (transitions). Let $N = (P, T, F)$ be a Petri net. Elements of $P \cup T$ are called *nodes*. A node x is an *input node* of another node y if and only if there is a directed arc from x to y (i.e., $(x, y) \in F$). Node x is an *output node* of y if and only if $(y, x) \in F$. For any $x \in P \cup T$, $\bullet x = \{y \mid (y, x) \in F\}$ and $x\bullet = \{y \mid (x, y) \in F\}$. In Fig. 3.2, $\bullet c1 = \{a, f\}$ and $c1\bullet = \{b, c\}$.

Definition 3.3 (Firing rule) Let (N, M) be a marked Petri net with $N = (P, T, F)$ and $M \in \mathbb{B}(P)$. Transition $t \in T$ is *enabled*, denoted $(N, M)[t\rangle$, if and only if

$\bullet t \leq M$. The *firing rule* $\_[\_\rangle\_ \subseteq \mathcal{N} \times T \times \mathcal{N}$ is the smallest relation satisfying for any $(N, M) \in \mathcal{N}$ and any $t \in T$, $(N, M)[t\rangle \Rightarrow (N, M)\ [t\rangle\ (N, (M \setminus \bullet t) \uplus t\bullet)$.

$(N, M)[t\rangle$ denotes that t is enabled at marking M, e.g., $(N, [start])[a\rangle$ in Fig. 3.2. $(N, M)\ [t\rangle\ (N, M')$ denotes that firing this enabled transition results in marking M'. For example, $(N, [start])\ [a\rangle\ (N, [c1, c2])$ and $(N, [c3, c4])\ [e\rangle\ (N, [c5])$.

Let (N, M_0) with $N = (P, T, F)$ be a marked Petri net. A sequence $\sigma \in T^*$ is called a *firing sequence* of (N, M_0) if and only if, for some natural number $n \in \mathbb{N}$, there exist markings $M_1, \ldots, M_n$ and transitions $t_1, \ldots, t_n \in T$ such that $\sigma = \langle t_1 \ldots t_n \rangle$ and, for all i with $0 \leq i < n$, $(N, M_i)[t_{i+1}\rangle$ and $(N, M_i)\ [t_{i+1}\rangle\ (N, M_{i+1})$.[1]

Let (N, M_0) be the marked Petri net shown in Fig. 3.2, i.e., $M_0 = [start]$. The empty sequence $\sigma = \langle\ \rangle$ is enabled in (N, M_0), i.e., $\langle\ \rangle$ is a firing sequence of (N, M_0). The sequence $\sigma = \langle a, b \rangle$ is also enabled and firing σ results in marking $[c2, c3]$. Another possible firing sequence is $\sigma = \langle a, c, d, e, f, b, d, e, g \rangle$. A marking M is *reachable* from the initial marking M_0 if and only if there exists a sequence of enabled transitions whose firing leads from M_0 to M. The set of reachable markings of (N, M_0) is denoted $[N, M_0\rangle$. The marked Petri net shown in Fig. 3.2 has seven reachable markings.

In Fig. 3.2, transitions are identified by a single letter, but also have a longer label describing the corresponding activity. Thus far we ignored these labels.

Definition 3.4 (Labeled Petri net) A *labeled Petri net* is a tuple $N = (P, T, F, A, l)$ where (P, T, F) is a Petri net as defined in Definition 3.2, $A \subseteq \mathscr{A}$ is a set of *activity labels*, and $l \in T \rightarrow A$ is a *labeling function*.

In principle, multiple transitions may bear the same label. One can think of the transition label as the *observable action*. Sometimes one wants to express that particular transitions are not observable. For this we reserve the label τ. A transition t with $l(t) = \tau$ is unobservable. Such transitions are often referred to as *silent* or *invisible*. It is easy to convert any Petri net into a labeled Petri net; just take $A = T$ and $l(t) = t$ for any $t \in T$. The reverse is not always possible, e.g., when several transitions have the same label. It is also possible to convert a marked (labeled) Petri net into a transition system as is shown next.

Definition 3.5 (Reachability graph) Let (N, M_0) with $N = (P, T, F, A, l)$ be a marked labeled Petri net. (N, M_0) defines a transition system $TS = (S, A', T')$ with $S = [N, M_0\rangle$, $S^{start} = \{M_0\}$, $A' = A$, and $T' = \{(M, l(t), M') \in S \times A \times S \mid \exists_{t \in T}\ (N, M)\ [t\rangle\ (N, M')\}$. TS is often referred to as the *reachability graph* of (N, M_0).

Figure 3.3 shows the transition system generated from the labeled marked Petri net shown in Fig. 3.2. States correspond to reachable markings, i.e., multi-sets of

[1] X^* is the set of sequences containing elements of X, i.e., for any $n \in \mathbb{N}$ and $x_1, x_2, \ldots, x_n \in X$: $\langle x_1, x_2, \ldots, x_n \rangle \in X^*$. See also Sect. 5.2.

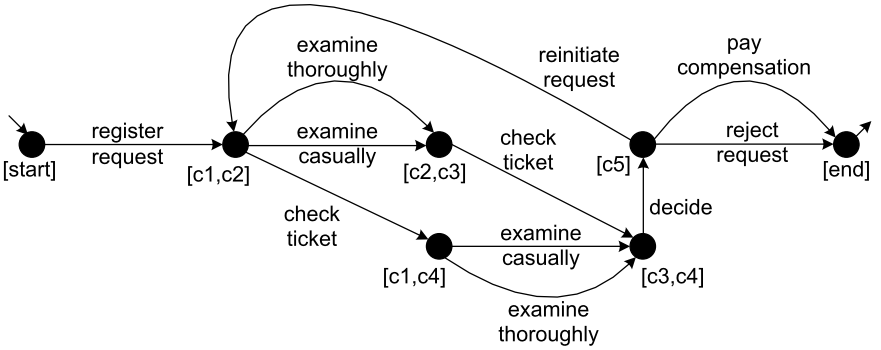


Fig. 3.3 The reachability graph of the marked Petri net shown in Fig. 3.2

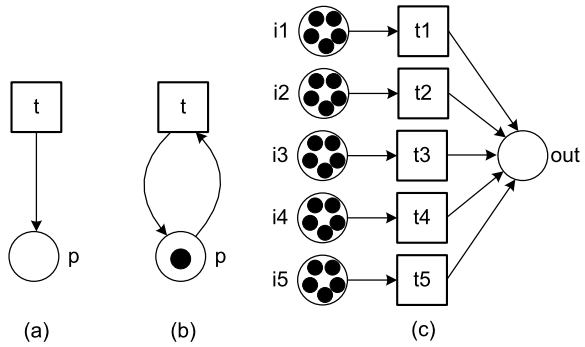


Fig. 3.4 Three Petri nets: (**a**) a Petri net with an infinite state space, (**b**) a Petri net with only one reachable marking, (**c**) a Petri net with 7776 reachable markings

tokens. Note that $S^{start} = \{[start]\}$ is a singleton containing the initial marking of the Petri net. The Petri net does not explicitly define a set of final markings S^{end}. However, in this case it is obvious to take $S^{end} = \{[end]\}$. Later, we will see that it is sometimes useful to distinguish deadlocks and livelocks from successful termination.

Note that we are overloading the term “transition”; the term may refer to a “box” in a Petri net or an “arc” in a transition system. In fact, one transition in a Petri net may correspond to many transitions in the corresponding transition system.

The Petri net in Fig. 3.2 and the transition system in Fig. 3.3 are of similar sizes. If the model contains a lot of concurrency or multiple tokens reside in the same place, then the transition system is much bigger than the Petri net. In fact, a marked Petri net may have infinitely many reachable states. The marked Petri net in Fig. 3.4(a) consists of only one place and one transition. Nevertheless, its corresponding transition system has infinitely many states: $S = \{[p^k] \mid k \in \mathbb{N}\}$. In this example, transition t is continuously enabled because it has no input place. Therefore, it can put any number of tokens in p. The Petri net in Fig. 3.4(b) has two arcs rather than one and now the only reachable state is $[p]$. The marked Petri net in Fig. 3.4(c) shows the effect of concurrency. The corresponding transition system has $6^5 = 7776$ states and 32,400 transitions.

Modern computers can easily compute reachability graphs with millions of states and analyze them. If the reachability graph is infinite, one can resort to the so-called *coverability graph* that presents a kind of over-approximation of the state space [117]. By constructing the reachability graph (if possible) or the coverability graph one can answer a variety of questions regarding the behavior of the process modeled. Moreover, dedicated analysis techniques can also answer particular questions without constructing the state space, e.g., using the linear-algebraic representation of the Petri net. It is outside the scope of this book to elaborate on these. However, we list some generic properties typically investigated in the context of a marked Petri net.

- A marked Petri net (N, M_0) is *k-bounded* if no place ever holds more that k tokens. Formally, for any $p \in P$ and any $M \in [N, M_0\rangle$: $M(p) \leq k$. The marked Petri net in Fig. 3.4(c) is 25-bounded because in none of the 7776 reachable markings there is a place with more than 25 tokens. It is not 24-bounded, because in the final marking place *out* contains 25 tokens.
- A marked Petri net is *safe* if and only if it is 1-bounded. The marked Petri net shown in Fig. 3.2 is safe because in each of the seven reachable markings there is no place holding multiple tokens.
- A marked Petri net is *bounded* if and only if there exists a $k \in \mathbb{N}$ such that it is k-bounded. Figure 3.4(a) shows an unbounded net. The two other marked Petri nets in Fig. 3.4 (i.e., (b) and (c)) are bounded.
- A marked Petri net (N, M_0) is *deadlock free* if at every reachable marking at least one transition is enabled. Formally, for any $M \in [N, M_0\rangle$ there exists a transition $t \in T$ such that $(N, M)[t\rangle$. Figure 3.4(c) shows a net that is not deadlock free because at marking $[out^{25}]$ no transition is enabled. The two other marked Petri nets in Fig. 3.4 are deadlock free.
- A transition $t \in T$ in a marked Petri net (N, M_0) is *live* if from every reachable marking it is possible to enable t. Formally, for any $M \in [N, M_0\rangle$ there exists a marking $M' \in [N, M\rangle$ such that $(N, M')[t\rangle$. A marked Petri net is live if each of its transitions is live. Note that a deadlock-free Petri net does not need to be live. For example, merge the nets (b) and (c) in Fig. 3.4 into one marked Petri net. The resulting net is deadlock free, but not live.

Petri nets have a strong theoretical basis and can capture concurrency well. Moreover, a wide range of powerful analysis techniques and tools exists [117]. Obviously, this succinct model has problems capturing data-related and time-related aspects. Therefore, various types of high-level Petri nets have been proposed. *Colored Petri nets* (CPNs) are the most widely used Petri-net based formalism that can deal with data-related and time-related aspects [82, 149]. Tokens in a CPN carry a data value and have a timestamp. The data value, often referred to as “color”, describes the properties of the object modeled by the token. The timestamp indicates the earliest time at which the token may be consumed. Transitions can assign a delay to produced tokens. This way waiting and service times can be modeled. A CPN may be hierarchical, i.e., transitions can be decomposed into subprocesses. This way large models can be structured. CPN Tools is a toolset providing support for the modeling and analysis of CPNs (www.cpntools.org).

3.2.3 Workflow Nets

When modeling business processes in terms of Petri nets, we often consider a subclass of Petri nets known as *WorkFlow nets* (WF-nets) [136, 168]. A WF-net is a Petri net with a dedicated source place where the process starts and a dedicated sink place where the process ends. Moreover, all nodes are on a path from source to sink.

Definition 3.6 (Workflow net) Let $N = (P, T, F, A, l)$ be a (labeled) Petri net and $\bar{t}$ a fresh identifier not in $P \cup T$. N is a *workflow net* (WF-net) if and only if (a) P contains an input place i (also called source place) such that $\bullet i = \emptyset$, (b) P contains an output place o (also called sink place) such that $o\bullet = \emptyset$, and (c) $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\}, A \cup \{\tau\}, l \cup \{(\bar{t}, \tau)\})$ is strongly connected, i.e., there is a directed path between any pair of nodes in $\bar{N}$.

$\bar{N}$ is referred to as the short-circuited net [136]. The unique sink place o is connected to the unique source place i in the resulting net.

Figure 3.2 shows an example of a WF-net with $i = start$ and $o = end$. None of the three Petri nets in Fig. 3.4 is a WF-net.

Why are WF-nets particularly relevant for business process modeling? The reason is that the process models used in the context of BPM describe the *life-cycle of cases* of a given kind. Examples of cases are insurance claims, job applications, customer orders, replenishment orders, patients, and credit applications. The process model is instantiated once for each case. Each of these process instances has a well-defined start ("case creation") and end ("case completion"). In-between these points, activities are conducted according to a predefined procedure. One model may be instantiated many times. For example, the process of handling insurance claims may be executed for thousands or even millions of claims. These instances can be seen as copies of the same WF-net, i.e., tokens of different cases are not mixed.

WF-nets are also a natural representation for process mining. There is an obvious relation between the firing sequences of a WF-net and the traces found in event logs. Note that one can only learn models based on examples. In the context of market basket analysis, i.e., finding patterns in what customers buy, one needs many examples of customers buying particular collections of products. Similarly, process discovery uses sequences of activities in which each sequence refers to a particular process instance. These can be seen as firing sequences of an unknown WF-net. Therefore, we will often focus on WF-nets. Recall that the α-algorithm discovered the WF-net in Fig. 2.6 using the set of traces shown in Table 2.2. Every trace corresponds to a case executed from begin to end.

Not every WF-net represents a correct process. For example, a process represented by a WF-net may exhibit errors such as deadlocks, activities that can never become active, livelocks, or garbage being left in the process after termination. Therefore, we define the following well-known correctness criterion [136, 168]:

Definition 3.7 (Soundness) Let $N = (P, T, F, A, l)$ be a WF-net with input place i and output place o. N is *sound* if and only if

- (*safeness*) $(N, [i])$ is safe, i.e., places cannot hold multiple tokens at the same time;
- (*proper completion*) for any marking $M \in [N, [i]\rangle$, $o \in M$ implies $M = [o]$;
- (*option to complete*) for any marking $M \in [N, [i]\rangle$, $[o] \in [N, M\rangle$; and
- (*absence of dead parts*) $(N, [i])$ contains no dead transitions (i.e., for any $t \in T$, there is a firing sequence enabling t).

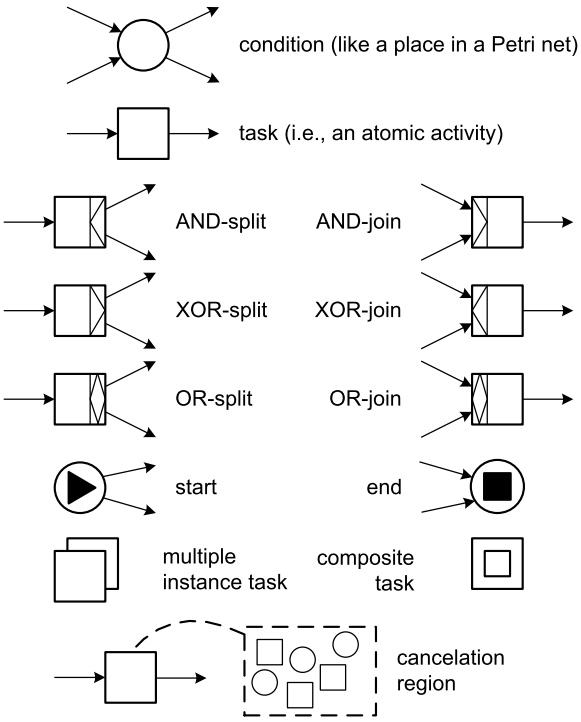
Note that the option to complete implies proper completion. The WF-net shown in Fig. 3.2 is sound. Soundness can be verified using standard Petri-net-based analysis techniques. In fact soundness corresponds to liveness and safeness of the corresponding short-circuited net $\bar{N}$ introduced in Definition 3.6 [136]. This way efficient algorithms and tools can be applied. An example of a tool tailored towards the analysis of WF-nets is *Woflan* [179]. This functionality is also embedded in our process mining tool *ProM* described in Sect. 11.3.

3.2.4 YAWL

YAWL is both a workflow modeling *language* and an open-source workflow *system* [132]. The acronym YAWL stands for “Yet Another Workflow Language”. The development of the YAWL language was heavily influenced by the *Workflow Patterns Initiative* [155, 191] mentioned earlier. Based on a systematic analysis of the constructs used by existing process modeling notations and workflow languages, a large collection of patterns was identified. These patterns cover all workflow perspectives, i.e., there are control-flow patterns, data patterns, resource patterns, change patterns, exception patterns, etc. The aim of YAWL is to offer direct support for many patterns while keeping the language simple. It can be seen as a reference implementation of the most important workflow patterns. Over time, the YAWL language and the YAWL system have increasingly become synonymous and have garnered widespread interest from both practitioners and the academic community alike. YAWL is currently one of the most widely used open-source workflow systems.

Here we restrict ourselves to the control-flow perspective. Figure 3.5 shows the main constructs. Each process has a dedicated start and end condition, like in WF-nets. Activities in YAWL are called *tasks*. *Conditions* in YAWL correspond to places in Petri nets. However, it is also possible to directly connect tasks without putting a condition in-between. Tasks have—depending on their type—a well-defined split and join semantics. An *AND-join/AND-split* task behaves like a transition, i.e., it needs to consume one token via each of the incoming arcs and produces a token along each of the outgoing arcs. An *XOR-split* selects precisely one of its outgoing arcs. The selection is based on evaluating data conditions. Only one token is produced and sent along the selected arc. An *XOR-join* is enabled once for every incoming token and does not need to synchronize. An *OR-split* selects one or more of its outgoing arcs. This selection is again based on evaluating data conditions. Note

Fig. 3.5 YAWL notation



that an OR-split may select 2 out of three 3 outgoing arcs. The semantics of the *OR-join* are more involved. The OR-join requires at least one input token, but also synchronizes tokens that are “on their way” to the OR-join. As long as another token may arrive via one of the ingoing arcs, the OR-join waits. YAWL also supports *cancelation regions*. A task may have a cancelation region consisting of conditions, tasks, and arcs. Once the task completes all tokens are removed from this region. Note that tokens for the task’s output conditions are produced after emptying the cancelation region. YAWL’s cancelation regions provide a powerful mechanism to abort work in parallel branches and to reset parts of the workflow. Tasks in a YAWL model can be *atomic* or *composite*. A composite task refers to another YAWL model. This way models can be structured hierarchically. Atomic and composite tasks can be instantiated multiple times in parallel. For example, when handling a customer order, some tasks needs to be executed for every order line. These order lines can be processed in any order. Therefore, a loop construct is less suitable. Figure 3.5 shows the icon for such a multiple instance task and all other constructs just mentioned.

Figure 3.6 shows an example YAWL model for the handling of a request for compensation within an airline. To show some of the features of YAWL, we extended the process described in Sect. 2.1 with some more complex behaviors. In the new model it is possible that both examinations are executed. By using an OR-split and an OR-join *examine causally* and/or *examine thoroughly* are executed. The model has also been extended with a cancelation region (see dotted box in Fig. 3.6). As

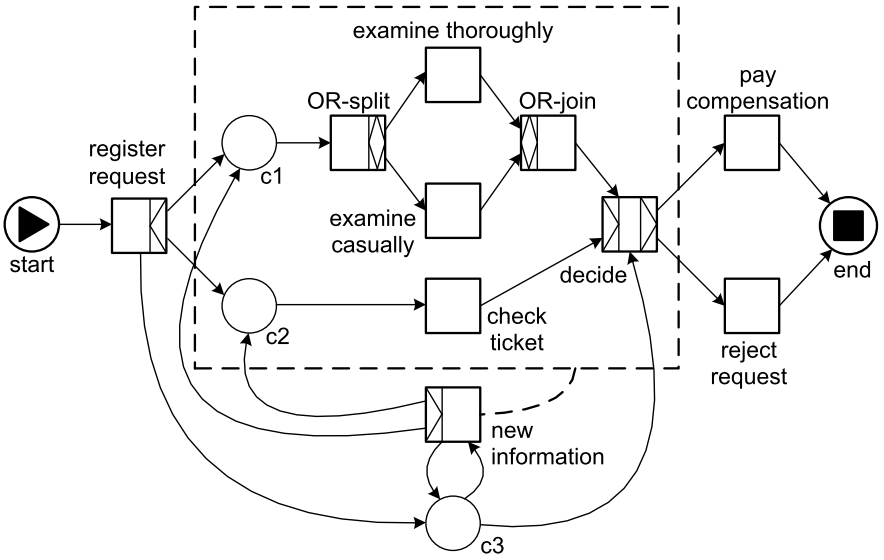


Fig. 3.6 Process model using the YAWL notation

long as there is a token in *c3*, task *new information* may be executed. When this happens, all tokens are removed from the region, i.e., checks and examinations are aborted. Task *new information* does not need to know where all tokens are and after the reset by this task the new state is $[c1, c2, c3]$. Explicit choices in YAWL (i.e., XOR/OR-splits) are driven by data conditions. In the Petri net in Fig. 3.2, all choices were non-deterministic. In the example YAWL model, the decision may be derived from the outcome of the check and the examination(s), i.e., the XOR-split *decide* may be based on data created in earlier tasks. As indicated, both the YAWL language and the YAWL system cover all relevant perspectives (resources, data, exceptions, etc.). For example, it is possible to model that decisions are taken by the manager and that it is not allowed that two examinations for the same request are done by the same person (4-eyes principle) [132].

3.2.5 Business Process Modeling Notation (BPMN)

Recently, the *Business Process Modeling Notation* (BPMN) has become one of the most widely used languages to model business processes. BPMN is supported by many tool vendors and has been standardized by the OMG [110]. Figure 3.7 shows the BPMN model already introduced in Sect. 2.1.

Figure 3.8 shows a small subset of all notational elements. Atomic activities are called *tasks*. Like in YAWL activities can be nested. Most of the constructs can be easily understood after the introduction to YAWL. A notable difference is that the

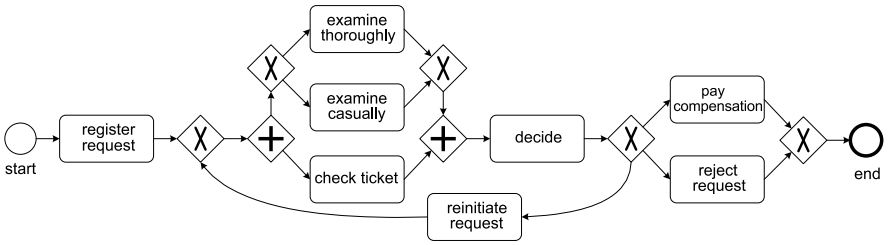
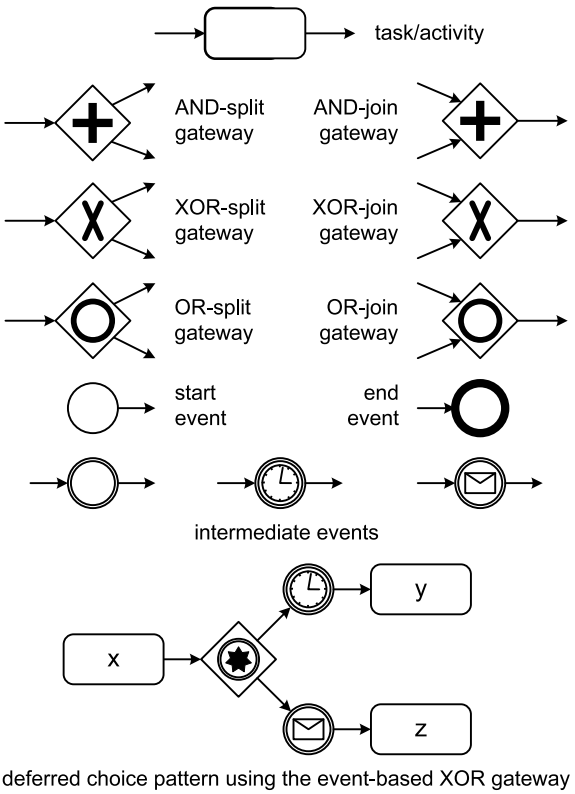


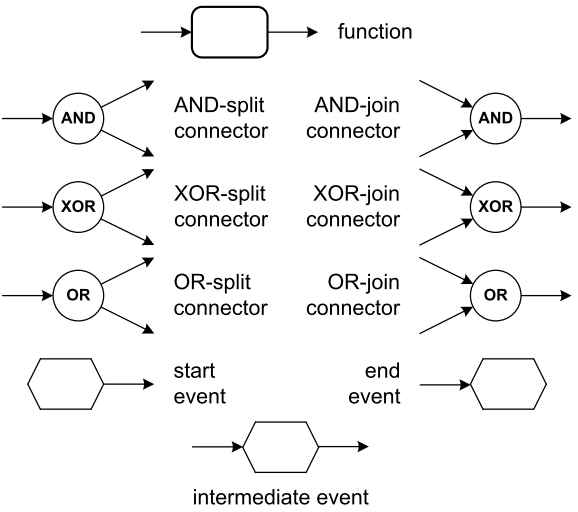
Fig. 3.7 Process model using the BPMN notation

Fig. 3.8 BPMN notation



routing logic is not associated with tasks but with separate *gateways*. Figure 3.8 shows that there are split and join gateways of different types: AND, XOR, OR. The splits are based on data conditions. An *event* is comparable to a place in a Petri net. However, the semantics of places in Petri nets and events in BPMN are quite different. There is no need to insert events in-between activities and events cannot have multiple input or output arcs. *Start events* have one outgoing arc, *intermediate events* have one incoming and one outgoing arc, and *end events* have one incoming arc. Unlike in YAWL or a Petri net, one cannot have events with multiple

Fig. 3.9 EPC notation



incoming or outgoing arcs; splitting and joining needs to be done using gateways. To model the so-called *deferred choice* workflow pattern [155] one needs to use the event-based XOR gateway shown in Fig. 3.8. This illustrates the use of events. After executing task x there is a race between two events. One of the events is triggered by a timeout. The other event is triggered by an external massage. The first event to occur determines the route taken. If the message arrives before the timer goes off, task z is executed. If the timer goes off before the massage arrives, task y is executed. Note that this construct can easily be modeled in YAWL using a condition with two output arcs.

Figure 3.8 shows just a tiny subset of all notations provided by BPMN. Most vendors support only a small subset of BPMN in their products. Moreover, users typically use only few BPMN constructs. In [193], it was shown that the average subset of BPMN used in real-life models consists of less than 10 different symbols (despite the more than 50 distinct graphical elements offered to the modeler). For this reason, we will be rather pragmatic when it comes to process models and their notation.

3.2.6 Event-Driven Process Chains (EPCs)

Event-driven Process Chains (EPCs) provide a classical notation to model business processes [126]. The notation is supported by products such as ARIS and SAP R/3. Basically, EPCs cover a limited subset of BPMN and YAWL while using a dedicated graphical notation.

Figure 3.9 provides an overview of the different notational elements. *Functions* correspond to activities. A function has precisely one input arc and one output arc. Therefore, splitting and joining can only be modeled using *connectors*. These are

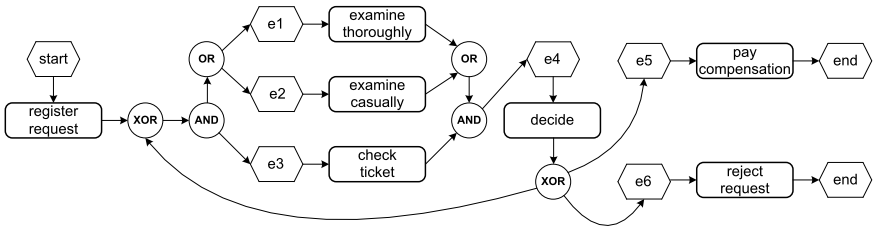


Fig. 3.10 Process model using the EPC notation

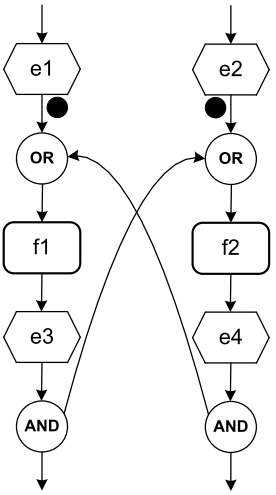


Fig. 3.11 The so-called “vicious circle” expressed using the EPC notation

comparable to the gateways in BPMN. Again splits and joins of type AND, XOR, and OR are supported. Like in BPMN there are three types of *events* (start, intermediate, and end). Events and functions need to alternate along any path, i.e., it is not allowed to connect events to events or functions to functions.

Figure 3.10 shows another variation of the process for handling a request for compensation. Note that, because of the two OR connectors, it is possible to do both examinations or just one.

The EPC notation was one of the first notations allowing for OR splits and joins. However, the people who developed and evangelized EPCs did not provide clear semantics nor some reference implementation [154]. This triggered lively debates resulting in various proposals and alternative implementations. Consider, for example, the so-called “vicious circle” shown in Fig. 3.11. The two tokens show the state of this process fragment; events *e1* and *e2* hold a token. It is unclear what could happen next, because both OR-joins depend on one another.

Should the OR-join below *e1* block or not? Suppose that this OR-join blocks, then by symmetry also the other OR-join following *e2* should block and the whole EPC deadlocks in the state shown Fig. 3.11. This seems to be wrong because if it deadlocks, the OR join will never receive an additional token and hence should

not have waited in the first place. Suppose that the OR-join following *e1* does not block. By symmetry the other OR-join should also not block and both *f1* and *f2* are executed and tokens flow towards both OR-joins via the two AND-splits. However, this implies that the OR-joins should both have blocked. Hence, there is a *paradox* because all possible decisions are wrong.

The vicious circle paradox shows that higher-level constructs may introduce all kinds of subtle semantic problems. Despite these problems and the different notations, the core concepts of the various languages are very similar.

3.2.7 Causal Nets

The notations discussed thus far connect activities (i.e., transitions, tasks, functions) through model elements like places (Petri nets), conditions (YAWL), connectors and events (EPC), gateways and events (BPMN). These elements interconnect activities but do not leave any “marks” in the event log, i.e., they need to be inferred by analyzing the behavior. Since the log does not provide concrete information about places, conditions, connectors, gateways and events, some mining algorithms use a representation consisting of just activities and no connecting elements [4, 12, 66, 183, 184].

Causal nets are a representation tailored towards process mining. A causal net is a graph where nodes represent activities and arcs represent causal dependencies. Each activity has a set of possible *input bindings* and a set of possible *output bindings*. Consider, for example, the causal net shown in Fig. 3.12. Activity a has only an empty input binding as this is the start activity. There are two possible output bindings: $\{b, d\}$ and $\{c, d\}$. This means that a is followed by either b and d, or c and d. Activity e has two possible input bindings ($\{b, d\}$ and $\{c, d\}$) and three possible output bindings ($\{g\}$, $\{h\}$, and $\{f\}$). Hence, e is preceded by either b and d, or c and d, and is succeeded by just g, h or f. Activity z is the end activity having two input bindings and one output binding (the empty binding). This activity has been added to create a unique end point. All executions commence with start activity a and finish with end activity z. As will be shown later, the causal net shown in Fig. 3.12 and the Petri net shown in Fig. 3.2 are trace equivalent, i.e., they both allow for the same set of traces. However, there are no places in the causal net; the routing logic is solely represented by the possible input and output bindings.

Definition 3.8 (Causal net) A *Causal net* (C-net) is a tuple $C = (A, a_i, a_o, D, I, O)$ where:

- $A \subseteq \mathscr{A}$ is a finite set of *activities*;
- $a_i \in A$ is the *start activity*;
- $a_o \in A$ is the *end activity*;
- $D \subseteq A \times A$ is the *dependency relation*,
- $AS = \{X \subseteq \mathscr{P}(A) \mid X = \{\emptyset\} \ \vee \ \emptyset \notin X\}$;[2]

[2] $\mathscr{P}(A) = \{A' \mid A' \subseteq A\}$ is the powerset of A. Hence, elements of AS are *sets of sets* of activities.

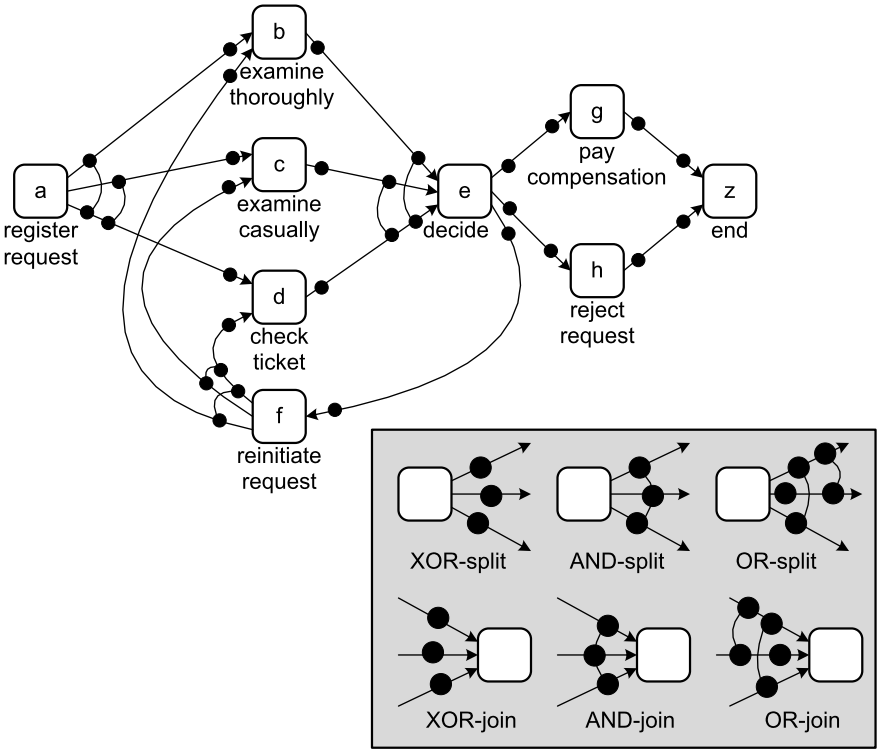


Fig. 3.12 Causal net C_1

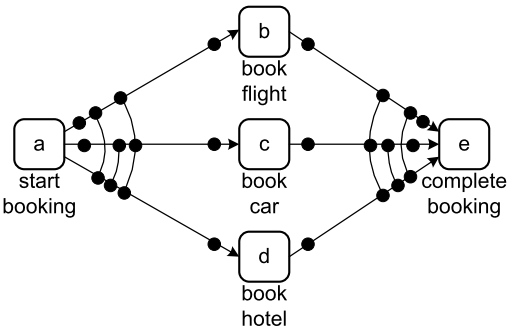
- $I \in A \rightarrow AS$ defines the set of possible *input bindings* per activity; and
- $O \in A \rightarrow AS$ defines the set of possible *output bindings* per activity,

such that

- $D = \{(a_1, a_2) \in A \times A \mid a_1 \in \bigcup_{as \in I(a_2)} as\}$;
- $D = \{(a_1, a_2) \in A \times A \mid a_2 \in \bigcup_{as \in O(a_1)} as\}$;
- $\{a_i\} = \{a \in A \mid I(a) = \{\emptyset\}\}$;
- $\{a_o\} = \{a \in A \mid O(a) = \{\emptyset\}\}$; and
- all activities in the graph (A, D) are on a path from a_i to a_o.

The C-net of Fig. 3.12 can be described as follows. $A = \{a, b, c, d, e, f, g, h, z\}$ is the set of activities, $a = a_i$ is the unique start activity, and $z = a_o$ is the unique end activity. The arcs shown in Fig. 3.12 visualize the dependency relation $D = \{(a, b), (a, c), (a, d), (b, e), \ldots, (g, z), (h, z)\}$. Functions I and O describe the sets of possible input and output bindings. $I(a) = \{\emptyset\}$ is the set of possible input bindings of a, i.e., the only input binding is the empty set of activities. $O(a) = \{\{b, d\}, \{c, d\}\}$ is the set of possible output bindings of a, i.e., activity a is followed by d and either b or c. $I(b) = \{\{a\}, \{f\}\}$, $O(b) = \{\{e\}\}$, $\ldots$,

Fig. 3.13 Causal net C_2



$I(z) = \{\{g\}, \{h\}\}$, $O(z) = \{\emptyset\}$. Note that any element of AS is a set of sets of activities, e.g., $\{\{b, d\}, \{c, d\}\} \in AS$. If one of the elements is the empty set, then there cannot be any other elements, i.e., for any $X \in AS$: $X = \{\emptyset\}$ or $\emptyset \notin X$. This implies that only the unique start activity a_i has the empty binding as (only) possible input binding. Similarly, only the unique end activity a_o has the empty binding as (only) possible output binding.

An *activity binding* is a tuple (a, as^I, as^O) denoting the occurrence of activity a with input binding as^I and output binding as^O. For example, $(e, \{b, d\}, \{f\})$ denotes the occurrence of activity e in Fig. 3.12 while being preceded by b and d, and succeeded by f.

Definition 3.9 (Binding) Let $C = (A, a_i, a_o, D, I, O)$ be a C-net. $B = \{(a, as^I, as^O) \in A \times \mathscr{P}(A) \times \mathscr{P}(A) \mid as^I \in I(a) \wedge as^O \in O(a)\}$ is the set of *activity bindings*. A *binding sequence* σ is a sequence of activity bindings, i.e., $\sigma \in B^*$.

A possible binding sequence for the C-net of Fig. 3.12 is $\langle (a, \emptyset, \{b, d\}), (b, \{a\}, \{e\}), (d, \{a\}, \{e\}), (e, \{b, d\}, \{g\}), (g, \{e\}, \{z\}), (z, \{g\}, \emptyset)\rangle$.

Figure 3.13 shows another C-net modeling the booking of a trip. After activity a (*start booking*) there are three possible activities: b (*book flight*), c (*book car*), and d (*book hotel*). The process ends with activity e (*complete booking*). $O(a) = I(e) = \{\{b\}, \{c\}, \{b, d\}, \{c, d\}, \{b, c, d\}\}$, $I(a) = O(e) = \{\emptyset\}$, $I(b) = I(c) = I(d) = \{\{a\}\}$, and $O(b) = O(c) = O(d) = \{\{e\}\}$. A possible binding sequence for the C-net of Fig. 3.12 is $\langle (a, \emptyset, \{b, d\}), (d, \{a\}, \{e\}), (b, \{a\}, \{e\}), (e, \{b, d\}, \emptyset)\rangle$, i.e., the scenario in which a flight and a hotel are booked. Note that Fig. 3.13 does not allow for booking just a hotel nor is it possible to just book a flight and a car.

A binding sequence is *valid* if a predecessor activity and successor activity always “agree” on their bindings. For a predecessor activity x and successor activity y we need to see the following “pattern”: $\langle \ldots, (x, \{\ldots\}, \{y, \ldots\}), \ldots, (y, \{x, \ldots\}, \{\ldots\}), \ldots\rangle$, i.e., the occurrence of activity x with y in its output binding needs to be followed by the occurrence of activity y and the occurrence of activity y with x in its input binding needs to be preceded by the occurrence of activity x. To formalize the notion of a valid sequence, we first define the notion of *state*.

Definition 3.10 (State) Let $C = (A, a_i, a_o, D, I, O)$ be a C-net. $S = \mathbb{B}(A \times A)$ is the *state space* of C. $s \in S$ is a *state*, i.e., a multi-set of pending *obligations*. Function $\psi \in B^* \to S$ is defined inductively: $\psi(\langle\,\rangle) = [\,]$ and $\psi(\sigma \oplus (a, as^I, as^O)) = (\psi(\sigma) \setminus (as^I \times \{a\})) \uplus (\{a\} \times as^O)$ for any binding sequence $\sigma \oplus (a, as^I, as^O) \in B^*$.[3] $\psi(\sigma)$ is the state after executing binding sequence σ.

Consider C-net C_1 shown in Fig. 3.12. Initially there are no pending "obligations", i.e., no output bindings have been enacted without having corresponding input bindings. If activity binding $(a, \emptyset, \{b, d\})$ occurs, then $\psi(\langle(a, \emptyset, \{b, d\})\rangle) = \psi(\langle\,\rangle) \setminus (\emptyset \times \{a\}) \uplus (\{a\} \times \{b, d\}) = [\,] \setminus [\,] \uplus [(a, b), (a, d)] = [(a, b), (a, d)]$. State $[(a, b), (a, d)]$ denotes the obligation to execute both b and d using input bindings involving a. Input bindings remove pending obligations whereas output bindings create new obligations.

A *valid sequence* is a binding sequence that (a) starts with start activity a_i, (b) ends with end activity a_o, (c) only removes obligations that are pending, and (d) ends without any pending obligations. Consider, for example, the valid sequence $\sigma = \langle(a, \emptyset, \{b, d\}), (d, \{a\}, \{e\}), (b, \{a\}, \{e\}), (e, \{b, d\}, \emptyset)\rangle$ for C-net C_2 in Fig. 3.13:

$$\psi(\langle\,\rangle) = [\,]$$
$$\psi(\langle(a, \emptyset, \{b, d\})\rangle) = [(a, b), (a, d)]$$
$$\psi(\langle(a, \emptyset, \{b, d\}), (d, \{a\}, \{e\})\rangle) = [(a, b), (d, e)]$$
$$\psi(\langle(a, \emptyset, \{b, d\}), (d, \{a\}, \{e\}), (b, \{a\}, \{e\})\rangle) = [(b, e), (d, e)]$$
$$\psi(\langle(a, \emptyset, \{b, d\}), (d, \{a\}, \{e\}), (b, \{a\}, \{e\}), (e, \{b, d\}, \emptyset)\rangle) = [\,]$$

Sequence σ indeed starts with start activity a, ends with end activity e, only removes obligations that are pending (i.e., for every input binding there was an earlier output binding), and ends without any pending obligations: $\psi(\sigma) = [\,]$.

Definition 3.11 (Valid) Let $C = (A, a_i, a_o, D, I, O)$ be a C-net and $\sigma = \langle(a_1, as_1^I, as_1^O), (a_2, as_2^I, as_2^O), \ldots, (a_n, as_n^I, as_n^O)\rangle \in B^*$ a binding sequence. σ is a *valid sequence* of C if and only if:

- $a_1 = a_i$, $a_n = a_o$, and $a_k \in A \setminus \{a_i, a_o\}$ for $1 < k < n$;
- $\psi(\sigma) = [\,]$; and
- for any prefix $\langle(a_1, as_1^I, as_1^O), (a_2, as_2^I, as_2^O), \ldots, (a_k, as_k^I, as_k^O)\rangle = \sigma' \oplus (a_k, as_k^I, as_k^O) \in pref(\sigma)$: $(as_k^I \times \{a_k\}) \leq \psi(\sigma')$.

$V(C)$ is the set of all valid sequences of C.

[3] $\sigma_1 \oplus \sigma_2$ is the concatenation of two sequences, e.g., $\langle a, b, c\rangle \oplus \langle d, e\rangle = \langle a, b, c, d, e\rangle$. It is also possible to concatenate a sequence and an element, e.g., $\langle a, b, c\rangle \oplus d = \langle a, b, c, d\rangle$. Recall that X^* is the set of all sequences containing elements of X and $\langle\,\rangle$ is the empty sequence. See also Sect. 5.2 for more notations for sequences.

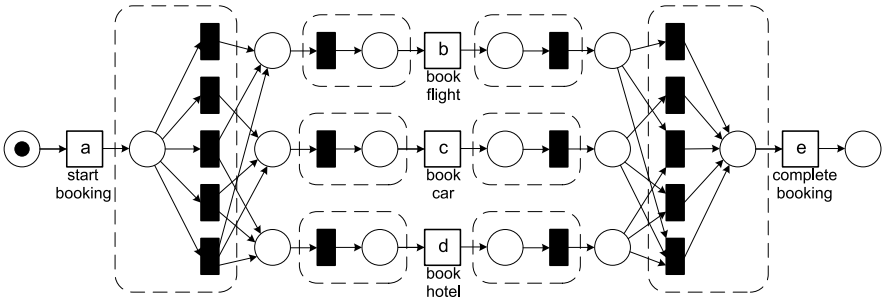


Fig. 3.14 A C-net transformed into a WF-net with silent transitions: every “sound run” of the WF-net corresponds to a valid sequence of the C-net C_2 shown in Fig. 3.13

The first requirement states that valid sequences start with a_i and end with a_o (a_i and a_o cannot appear in the middle of valid sequence). The second requirement states that at the end there should not be any pending obligations. (One can think of this as the constraint that no tokens left in the net.) The last requirement considers all non-empty prefixes of $\sigma, \langle (a_1, as_1^I, as_1^O), (a_2, as_2^I, as_2^O), \ldots, (a_k, as_k^I, as_k^O)\rangle$. The last activity binding of the prefix (i.e., (a_k, as_k^I, as_k^O)) should only remove pending obligations, i.e., $(as_k^I \times \{a_k\}) \leq \psi(\sigma')$ where $as_k^I \times \{a_k\}$ are the obligations to be removed and $\psi(\sigma')$ are the pending obligations just before the occurrence of the k-th binding. (One can think of this as the constraint that one cannot consume tokens that have not been produced.)

Figure 3.13 has 12 valid sequences: only b is executed ($\langle (a, \emptyset, \{b\}), (b, \{a\}, \{e\}), (e, \{b\}, \emptyset)\rangle$), only c is executed (besides a and e), b and d are executed (two possibilities), c and d are executed (two possibilities), and b, c and d are executed ($3! = 6$ possibilities). The C-net in Fig. 3.12 has infinitely many valid sequences because of the loop construct involving f. For example, $\langle (a, \emptyset, \{c, d\}), (c, \{a\}, \{e\}), (d, \{a\}, \{e\}), (e, \{c, d\}, \{f\}), (f, \{e\}, \{c, d\}), (c, \{f\}, \{e\}), (d, \{f\}, \{e\}), (e, \{c, d\}, \{g\}), (g, \{e\}, \{z\}), (z, \{g\}, \emptyset)\rangle$.

For the semantics of a C-net we only consider valid sequences, i.e., *invalid sequences are not part of the behavior* described by the C-net. This means that C-nets do not use plain “token-game like semantics” as in BPMN, Petri nets, EPCs, and YAWL. The semantics of C-nets are more declarative as they are defined over complete sequences rather than a local firing rule. This is illustrated by the WF-net shown in Fig. 3.14. This WF-net aims to model the semantics of the C-net C_2 in Fig. 3.13. The input and output bindings are modeled by *silent transitions*. In Fig. 3.14, these are denoted by black rectangles without labels. Note that the WF-net also allows for many invalid sequences. For example, it is possible to enable b, c and d. After firing b it is possible to fire e without firing c and d. This firing sequence does not correspond to a valid sequence because there are still pending commitments when executing the end activity e. However, if we only consider firing sequences of the WF-net that start with a token in the source place and end with a token in the sink place, then these match one-to-one with the valid sequences in $V(C_2)$.

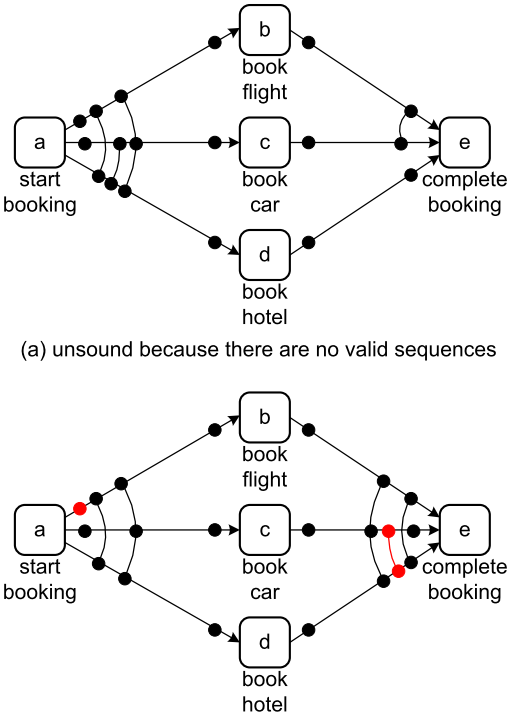


Fig. 3.15 Two C-nets that are not sound. The first net does not allow for any valid sequence, i.e., $V(C) = \emptyset$. The second net has valid sequences but also shows input/output bindings that are not realizable

The C-net shown in Fig. 3.12 and the WF-net shown in Fig. 3.2 are trace equivalent. Recall that in this comparison we consider all possible firing sequences of the WF-net and only valid sequences for the C-net.

We defined the notion of soundness for WF-nets (Definition 3.7) to avoid process models that have deadlocks, livelocks, and other anomalies. A similar notion can be defined for C-nets.

Definition 3.12 (Soundness of C-nets) A C-net $C = (A, a_i, a_o, D, I, O)$ is *sound* if (a) for all $a \in A$ and $as^I \in I(a)$ there exists a $\sigma \in V(C)$ and $as^O \subseteq A$ such that $(a, as^I, as^O) \in \sigma$, and (b) for all $a \in A$ and $as^O \in O(a)$ there exists a $\sigma \in V(C)$ and $as^I \subseteq A$ such that $(a, as^I, as^O) \in \sigma$.

Since the semantics of C-nets already enforce “proper completion” and the “option to complete”, we only need to make sure that there are valid sequences and that all parts of the C-net can potentially be activated by such a valid sequence. The C-nets C_1 and C_2 in Figs. 3.12 and 3.13 are sound. Figure 3.15 shows two C-nets that are not sound. In Fig. 3.15(a), there are no valid sequences because the output bindings of a and the input bindings of e do not match. For example, consider the binding sequence $\sigma = \langle (a, \emptyset, \{b\}), (b, \{a\}, \{e\}) \rangle$. Sequence σ cannot be extended into a valid sequence because $\psi(\sigma) = [(b, e)]$ and $\{b\} \notin I(e)$, i.e., the input bind-

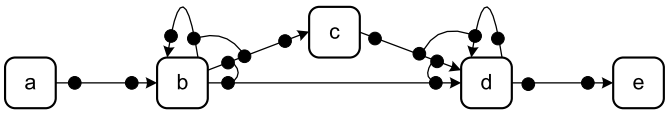


Fig. 3.16 A sound C-net that has no corresponding WF-net

ings of e do not allow for just booking a flight whereas the output bindings of a do. In Fig. 3.15(b), there are valid sequences, e.g., $\langle (a, \emptyset, \{c\}), (c, \{a\}, \{e\}), (e, \{c\}, \emptyset)\rangle$. However, not all bindings appear in one or more valid sequences. For example, the output binding $\{b\} \in O(a)$ does not appear in any valid sequence, i.e., after selecting just a flight the sequence cannot be completed properly. The input binding $\{c, d\} \in I(e)$ also does not appear in any valid sequence, i.e., the C-net suggests that only a car and hotel can be booked but there is no corresponding valid sequence.

Figure 3.16 shows an example of a sound C-net. One of the valid binding sequences for this C-net is $\langle (a, \emptyset, \{b\}), (b, \{a\}, \{b, c\}), (b, \{b\}, \{c, d\}), (c, \{b\}, \{d\}), (c, \{b\}, \{d\}), (d, \{b, c\}, \{d\}), (d, \{c, d\}, \{e\}), (e, \{d\}, \emptyset)\rangle$, i.e., the sequence $\langle a, b, b, c, c, d, d, e\rangle$. This sequence covers all the bindings. Therefore, the C-net is sound. Examples of other valid sequences are $\langle a, b, c, d, e\rangle$, $\langle a, b, c, b, c, d, d, e\rangle$, and $\langle a, b, b, b, c, c, c, d, d, d, e\rangle$. Figure 3.16 illustrates the expressiveness of C-nets. Note that there is no sound WF-net that reproduces exactly the set of valid sequences of this C-net. If we use the construction shown in Fig. 3.14 for the C-net of Fig. 3.16, we get a WF-net that is able to simulate the valid sequences. However, the resulting WF-net also allows for invalid behavior and it is impossible modify the model such that the set of firing sequences coincides with the set of valid sequences.

Causal nets are particularly suitable for process mining given their declarative nature and expressiveness without introducing all kinds of additional model elements (places, conditions, events, gateways, etc.). Several process discovery and conformance checking approaches use a similar representation [4, 12, 66, 183, 184]. In Chap. 7, we elaborate on this when discussing some of the more advanced process mining algorithms.

3.2.8 Process Trees

Petri nets, WF-nets, BPMN models, EPCs, YAWL models, and UML activity diagrams may suffer from deadlocks, livelocks, and other anomalies. Models having undesirable properties *independent* of the event log are called *unsound*. One does not need to look at the event log to see that an unsound model cannot describe the observed behavior well. Process discovery approaches using any of the graph-based process notations mentioned may produce unsound models. In fact, the majority of models in the search space tend to be unsound. This complicates discovery. C-nets address this problem by using more relaxed semantics. It is also possible to use

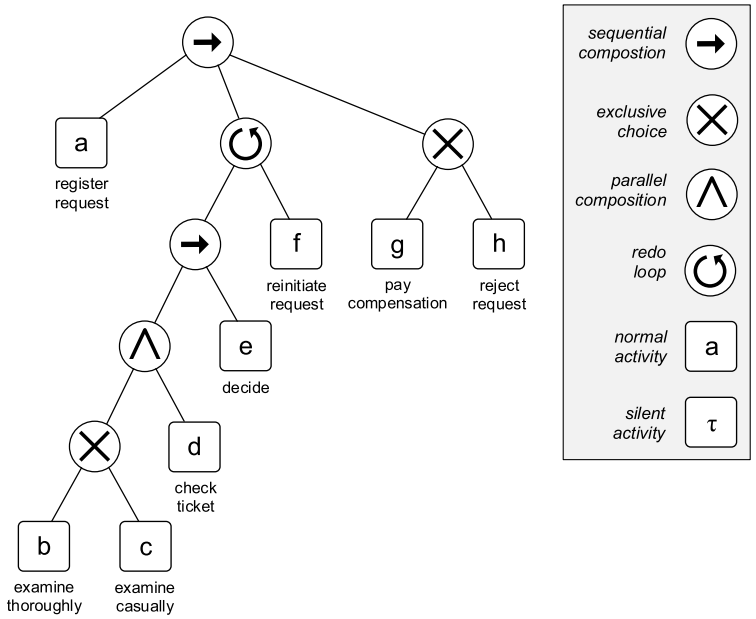


Fig. 3.17 Process tree $\rightarrow(a, \circlearrowleft(\rightarrow(\wedge(\times(b, c), d), e), f), \times(g, h))$ showing the different process tree operators

block-structured models that are sound by construction. In this section, we introduce *process trees* as a notation to represent such block-structured models. A process tree is a hierarchical process model where the (inner) nodes are operators such as sequence and choice and the leaves are activities.

Process trees are tailored towards process discovery. A range of *inductive process discovery* techniques exists for process trees [88–91]. These techniques benefit from the fact that the representation ensures soundness. The family of inductive mining techniques has variants that can handle infrequent behavior and deal with huge models and logs while ensuring formal correctness criteria such as the ability to rediscover the original model (see Sect. 7.5). Also the ETM (Evolutionary Tree Miner) approach described in [26] exploits the process tree representation. The fact that the search space is limited to sound models is a key ingredient of this highly flexible genetic process mining approach.

Figure 3.17 shows a *process tree* modeling the handling of a request for compensation within an airline. The set of traces that can be generated by this model is identical to the traces generated by the WF-net in Fig. 3.2 (the two models are trace equivalent). The inner nodes of the process tree represent operators. The leaves represent activities. There is one root node. Figure 3.17 shows the four types of operators can be used in a process tree: $\rightarrow$ (sequential composition), $\times$ (exclusive choice), $\wedge$ (parallel composition), and $\circlearrowleft$ (redo loop).

A sequence operator executes its children in sequential order. Activity a is the first child of the root node in Fig. 3.17. Since this node is a sequence node, every

process instance starts with activity a followed by the subtree starting with the redo loop ($\circlearrowleft$). After this subtree in the middle, the rightmost subtree is executed. The latter subtree models a choice ($\times$) between g and h.

The process tree in Fig. 3.17 can also be represented textually:

$$\rightarrow\big(a, \circlearrowleft\big(\rightarrow\big(\wedge\big(\times(b,c),d\big),e\big),f\big),\times(g,h)\big)$$

The rightmost subtree modeling the choice between activities g and h is represented as $\times(g,h)$. The redo loop $\circlearrowleft(\rightarrow(\wedge(\times(b,c),d),e),f)$ starts with its leftmost child and may loop back through any of its other children. In the process tree of Fig. 3.17, it is possible to loop back via “redo” activity f. The leftmost child (“do part”) is $\rightarrow(\wedge(\times(b,c),d),e)$, i.e., a sequence that ends with activity e which is preceded by the subtree $\wedge(\times(b,c),d)$ where activity d is executed in parallel with a choice between b and c. The subtree $\wedge(\times(b,c),d)$ has four potential behaviors: $\langle b,d\rangle$, $\langle c,d\rangle$, $\langle d,b\rangle$, and $\langle d,c\rangle$.

The same activity may appear multiple times in the same process tree. For example, process tree $\rightarrow(a,a,a)$ models a sequence of three a activities. From a behavioral point of view, $\rightarrow(a,a,a)$ and $\wedge(a,a,a)$ are indistinguishable. Both have one possible trace, $\langle a,a,a\rangle$.

A silent activity is denoted by τ and cannot be observed. Process tree $\times(a,\tau)$ can be used to model an activity a that can be skipped. Process tree $\circlearrowleft(a,\tau)$ can be used to model the process that executes a at least once. The “redo” part is silent, so the process can loop back without executing any activity. Process tree $\circlearrowleft(\tau,a)$ models a process that executes a any number of times. The “do” part is now silent and activity a is in the “redo” part. This way it is also possible to not execute a at all. The smallest process tree is a tree consisting of just one activity. In this case the root node is also a leaf node and there are no operator nodes.

Definition 3.13 (Process tree) Let $A \subseteq \mathscr{A}$ be a finite set of activities with $\tau \notin A$. $\bigoplus = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ is the set of *process tree operators*.

- If $a \in A \cup \{\tau\}$, then $Q = a$ is a process tree,
- If $n \geq 1$, $Q_1, Q_2, \ldots, Q_n$ are process trees, and $\oplus \in \{\rightarrow, \times, \wedge\}$, then $Q = \oplus(Q_1, Q_2, \ldots Q_n)$ is a process tree, and
- If $n \geq 2$ and $Q_1, Q_2, \ldots, Q_n$ are process trees, then $Q = \circlearrowleft(Q_1, Q_2, \ldots Q_n)$ is a process tree.

$\mathscr{Q}_A$ is the set of *all process trees* over A.

The redo loop operator $\circlearrowleft$ has at least two children. The first child is the “do” part and the other children are “redo” parts. Process tree $\circlearrowleft(a,b,c)$ allows for traces $\{\langle a\rangle, \langle a,b,a\rangle, \langle a,c,a\rangle, \langle a,b,a,b,a\rangle, \langle a,c,a,c,a\rangle, \langle a,c,a,b,a\rangle, \langle a,b,a,c,a\rangle, \ldots\}$. Activity a is executed at least once and the process always starts and ends with a. The “do” part alternates with the “redo” parts b or c. When looping back either b or c is executed.

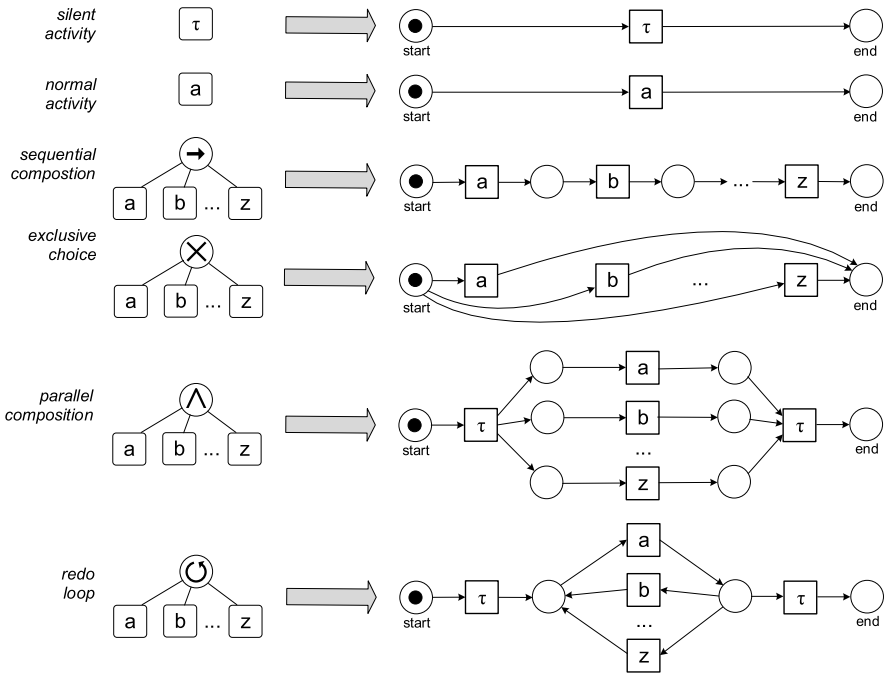


Fig. 3.18 Mapping process trees onto WF-nets

The redo loop operator $\circlearrowleft$ is often used in conjunction with silent activity τ. For example, $\circlearrowleft(\tau, a, b, c, \ldots, z)$ allows for any “word” involving activities $a, b, c, \ldots, z$. Example traces are $\langle\,\rangle$, $\langle a, b, b, a\rangle$, and $\langle w, o, r, d\rangle$.

Process trees can be converted to WF-nets as shown in Fig. 3.18. A silent activity is mapped onto a transition having a τ label. The mappings for $\rightarrow$ (sequential composition), $\times$ (exclusive choice), and $\wedge$ (parallel composition) are fairly straightforward. Silent transitions are used to model the start and end of the parallel composition. This is done to preserve the WF-net structure. The redo loop ($\circlearrowleft$) has one “do” part (activity a in Fig. 3.18) and one or more “redo” parts (activities b until z in Fig. 3.18). The direction of the arcs in the Petri net show the difference in semantics between the “do” and “redo” parts. Silent transitions are used to model the entry and exit of the redo loop. The mapping in Fig. 3.18 can be applied recursively and used to transform any process tree into a *sound* WF-net.

The mapping in Fig. 3.18 can easily be adapted for other representations such as BPMN, YAWL, EPCs, UML activity diagrams, statecharts, etc. The structured nature of process trees makes the conversion to other modeling notations straightforward. Conversions in the other direction (for example, from non-block-structured models to process trees) are more involved, but also less relevant since we only use process trees for process discovery. The mapping from process trees to WF-nets allows us to use existing conformance checking and performance analysis techniques.

The semantics of process trees can also be defined directly (without a mapping to WF-nets). To do this we first define two operators on sequences, concatenation ($\cdot$) and shuffle ($\diamond$).

Let $\sigma_1, \sigma_2 \in A^*$ be two sequences over A. $\sigma_1 \cdot \sigma_2 \in A^*$ *concatenates* two sequences, e.g., $\langle w, o \rangle \cdot \langle r, d \rangle = \langle w, o, r, d \rangle$. Concatenation can be generalized to sets of sequences. Let $S_1, S_2, \ldots, S_n \subseteq A^*$ be sets of sequences over A. $S_1 \cdot S_2 = \{\sigma_1 \cdot \sigma_2 \mid \sigma_1 \in S_1 \ \wedge \ \sigma_2 \in S_2\}$. For example, $\{\langle w, o \rangle, \langle\ \rangle\} \cdot \{\langle r, d \rangle, \langle k \rangle\} = \{\langle w, o, r, d \rangle, \langle w, o, k \rangle, \langle r, d \rangle, \langle k \rangle\}$. $\bigodot_{1 \leq i \leq n} \ S_i = S_1 \cdot S_2 \cdots S_n$ concatenates an ordered collection of sets of sequences.

$\sigma_1 \diamond \sigma_2$ generates the set of all interleaved sequences (shuffle). For example, $\langle w, o \rangle \diamond \langle r, d \rangle = \{\langle w, o, r, d \rangle, \langle w, r, o, d \rangle, \langle r, w, o, d \rangle, \langle w, r, d, o \rangle, \langle r, w, d, o \rangle, \langle r, d, w, o \rangle\}$. Note that the ordering in the original sequences is preserved, e.g., d cannot appear before r. Another example is $\langle w, o, r \rangle \diamond \langle d \rangle = \{\langle w, o, r, d \rangle, \langle w, o, d, r \rangle, \langle w, d, o, r \rangle, \langle d, w, o, r \rangle\}$. The shuffle operator can also be generalized to sets of sequences. $S_1 \diamond S_2 = \{\sigma \in \sigma_1 \diamond \sigma_2 \mid \sigma_1 \in S_1 \ \wedge \ \sigma_2 \in S_2\}$. The shuffle operator is commutative and associative, i.e., $S_1 \diamond S_2 = S_2 \diamond S_1$ and $(S_1 \diamond S_2) \diamond S_3 = S_1 \diamond (S_2 \diamond S_3)$. We write $\diamondsuit_{1 \leq i \leq n} \ S_i = S_1 \diamond S_2 \diamond \cdots \diamond S_n$ to interleave sets of sequences.

Definition 3.14 (Semantics) Let $Q \in \mathcal{Q}_A$ be a process tree over A. $\mathcal{L}(Q)$ is the *language* of Q, i.e., the set of traces that can be generated by it. $\mathcal{L}(Q)$ is defined recursively:

- $\mathcal{L}(Q) = \{\langle a \rangle\}$ if $Q = a \in A$,
- $\mathcal{L}(Q) = \{\langle\ \rangle\}$ if $Q = \tau$,
- $\mathcal{L}(Q) = \bigodot_{1 \leq i \leq n} \ \mathcal{L}(Q_i)$ if $Q = \rightarrow(Q_1, Q_2, \ldots, Q_n)$,
- $\mathcal{L}(Q) = \bigcup_{1 \leq i \leq n} \ \mathcal{L}(Q_i)$ if $Q = \times(Q_1, Q_2, \ldots, Q_n)$,
- $\mathcal{L}(Q) = \diamondsuit_{1 \leq i \leq n} \ \mathcal{L}(Q_i)$ if $Q = \wedge(Q_1, Q_2, \ldots, Q_n)$,
- $\mathcal{L}(Q) = \{\sigma_1 \cdot \sigma'_1 \cdot \sigma_2 \cdot \sigma'_2 \cdots \sigma_m \in A^* \mid m \geq 1 \ \wedge \ \forall_{1 \leq j \leq m} \ \sigma_j \in \mathcal{L}(Q_1) \ \wedge \ \forall_{1 \leq j < m} \ \sigma'_j \in \bigcup_{2 \leq i \leq n} \ \mathcal{L}(Q_i)\}$ if $Q = \circlearrowleft(Q_1, Q_2, \ldots, Q_n)$.

The following examples further illustrate the process tree operators and their semantics:

- $\mathcal{L}(\tau) = \{\langle\ \rangle\}$,
- $\mathcal{L}(a) = \{\langle a \rangle\}$,
- $\mathcal{L}(\rightarrow(a, b, c)) = \{\langle a, b, c \rangle\}$,
- $\mathcal{L}(\times(a, b, c)) = \{\langle a \rangle, \langle b \rangle, \langle c \rangle\}$,
- $\mathcal{L}(\wedge(a, b, c)) = \{\langle a, b, c \rangle, \langle a, c, b \rangle, \langle b, a, c \rangle, \langle b, c, a \rangle, \langle c, a, b \rangle, \langle c, b, a \rangle\}$,
- $\mathcal{L}(\circlearrowleft(a, b, c)) = \{\langle a \rangle, \langle a, b, a \rangle, \langle a, c, a \rangle, \langle a, b, a, c, a \rangle, \langle a, c, a, b, a \rangle, \ldots\}$,
- $\mathcal{L}(\rightarrow(a, \times(b, c), \wedge(a, a))) = \{\langle a, b, a, a \rangle, \langle a, c, a, a \rangle\}$,
- $\mathcal{L}(\times(\tau, a, \tau, \rightarrow(\tau, b), \wedge(c, \tau))) = \{\langle\ \rangle, \langle a \rangle, \langle b \rangle, \langle c \rangle\}$, and
- $\mathcal{L}(\circlearrowleft(a, \tau, c)) = \{\langle a \rangle, \langle a, a \rangle, \langle a, a, a \rangle, \langle a, c, a \rangle, \langle a, a, c, a \rangle, \langle a, c, a, c, a \rangle, \ldots\}$.

Process trees are *sound by construction*. Process discovery algorithms may exploit this when searching for a process model describing the event data. There are some similarities with other notations. *Process calculi* such as CSP and CCS use similar operators to model processes. Process trees can be viewed as a carefully

chosen subset. *Regular expressions* can model regular languages, e.g., $a^*(b|c)d^*$ denotes the set of traces starting with zero or more a's, followed by b or c, followed by zero or more d's. Process trees are in-between process calculi and regular expressions, and are tailored towards process discovery. Process calculi can handle concurrency, but are difficult to discover from event data (unless a similar subset is chosen). Regular expressions do not provide operators for concurrency and redo loops. However, in terms of expressiveness, process trees are comparable to regular expressions. Process trees are also related to *soundness preserving reduction rules* for Petri nets [168]. Reductions rules are normally used to reduce the size of a Petri net while preserving essential properties (e.g., soundness, liveness, boundedness, etc.). Starting from a WF-net with one transition, they can also be applied in reverse direction to produce larger sound WF-nets.

Section 7.5 introduces inductive process discovery techniques. Then the rationale for the choice of operators will become clearer. For example, $↻(\tau, a, b, c, \ldots, z)$ will be used as a last resort when all other operators are not applicable.

3.3 Model-Based Process Analysis

In Sect. 2.1, we discussed the different reasons for making models. Figure 2.4 illustrated the use of these models in the BPM life-cycle. Subsequent analysis showed that existing approaches using process models ignore event data. In later chapters we will show how to exploit event data when analyzing processes and their models. However, before doing so, we briefly summarize mainstream approaches for model-based analysis: *verification* and *performance analysis*. Verification is concerned with the correctness of a system or process. Performance analysis focuses on flow times, waiting times, utilization, and service levels.

3.3.1 Verification

In Sect. 3.2.3, we introduced the notion of soundness for WF-nets. This is a correctness criterion that can be checked using verification techniques. Consider, for example, the WF-net shown in Fig. 3.19. The model has been extended to model that *check ticket* should wait for the completion of *examine casually* but not for *examine thoroughly*. Therefore, place *c6* was added to model this dependency. However, a modeling error was made. One of the requirements listed in Definition 3.7, i.e., the "option to complete" requirement, is not satisfied. The marking $[c2, c3]$ is reached by executing the firing sequence $\langle a, b \rangle$ and from this marking the desired end marking $[end]$ is no longer reachable. Note that $[c2, c3]$ is a dead marking, e.g., d is not enabled because $c6$ is empty.

Definition 3.12 defines a soundness notion for C-nets. The notion of soundness can easily be adapted for other languages such as YAWL, EPCs, and BPMN. When

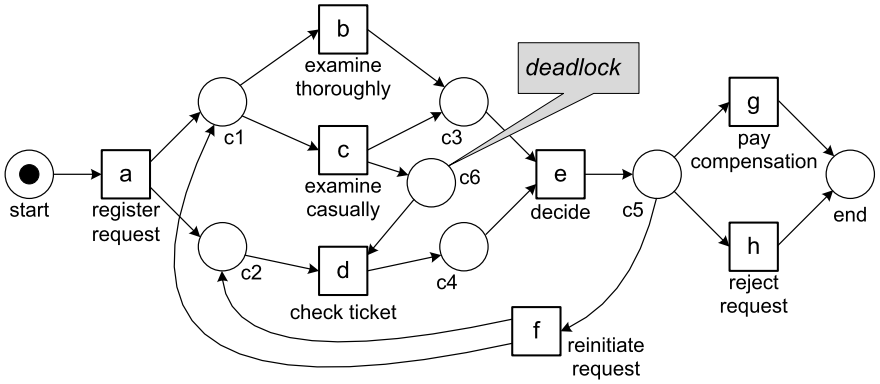


Fig. 3.19 A WF-net that is not sound

defining transition systems we already mentioned $S^{end} \subseteq S$ as the set of acceptable final states. Hence, we can define soundness as follows: a transition system is sound if and only if from any reachable state it is possible to reach a state in S^{end}. When introducing Petri nets we also defined generic properties such as liveness and boundedness. Some of these properties can be analyzed without constructing the state space. For example, for free-choice Petri nets, i.e., processes where choice and synchronization can be separated, liveness and boundedness can be checked by analyzing the rank of the corresponding incidence matrix [45]. Hence, soundness can be checked in polynomial time for free-choice WF-nets. Invariants can often be used to show boundedness or the unreachability of a particular marking. However, most of the more interesting verification questions require the exploration of (a part of) the state space.

Soundness is a generic property. Sometimes a more specific property needs to be investigated, e.g., “the ticket was checked for all rejected requests”. Such properties can be expressed in *temporal logic* [30, 93]. *Linear Temporal Logic* (LTL) is an example of a temporal logic that, in addition to classical logical operators, uses temporal operators such as: always ($\Box$), eventually ($\Diamond$), until ($\sqcup$), weak until (W), and next time ($\bigcirc$). The expression $\Diamond h \Rightarrow \Diamond d$ means that for all cases in which h (*reject request*) is executed also d (*check ticket*) is executed. Another example is $\Box(f \Rightarrow \Diamond e)$ that states that any occurrence of f will be followed by e. *Model checking* techniques can be used to check such properties [30].

Another verification task is the comparison of two models. For example, the implementation of a process needs is compared to the high-level specification of the process. As indicated before, there exist different equivalence notions (trace equivalence, branching bisimilarity, etc.) [176]. Moreover, there are also various simulation notions demanding that one model can “follow all moves” of the other but not vice versa (see also Sect. 6.3).

There are various tools to verify process models. A classical example is Woflan that is tailored towards checking soundness [179]. Also workflow systems such as YAWL provide verification capabilities. Consider, for example, the screenshot

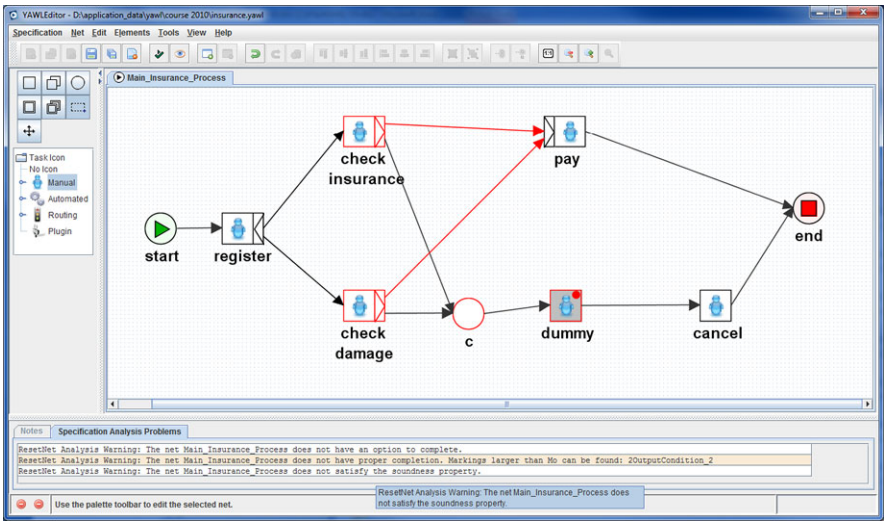


Fig. 3.20 An incorrect YAWL model: the cancelation region of *dummy* comprises of *check insurance*, *check damage*, condition *c* and the two implicit input conditions of *pay*. Hence, after cancelation, a token may be left on one of the output arcs of *register*

shown in Fig. 3.20. The figure shows the editor of YAWL while analyzing the model depicted. The process starts with task *register*. After this task, two checks can be done in parallel: *check insurance* and *check damage*. These tasks are XOR-splits; depending on the result of the check, one of the output arcs is selected. If both checks are OK, task *pay* is executed. If one of the checks indicates a problem, then the *dummy* task is executed. This task has a cancelation region consisting of *check insurance*, *check damage*, condition *c* and the two implicit input conditions of *pay*. The goal of this region is to remove all tokens, cancel the claim, and then end. However, the verifier of YAWL reports a problem. The YAWL model is not correct, because there may a be token pending in one of the implicit output conditions of *register*, i.e., there may be still a token on the arc connecting *register* and *check insurance* or on the arc connecting *register* and *check damage*. As a result the model may deadlock and “garbage” may be left behind. When these two implicit conditions are included in the cancelation region of the *dummy* task, then the verifier of YAWL will not find any problems and the model is indeed free of deadlocks and other anomalies.

3.3.2 Performance Analysis

The performance of a process or organization can be defined in different ways. Typically, three dimensions of performance are identified: *time*, *cost* and *quality*. For each of these performance dimensions different *Key Performance Indicators* (KPIs)

can be defined. When looking at the *time dimension* the following performance indicators can be identified:

- The *lead time* (also referred to as flow time) is the total time from the creation of the case to the completion of the case. In terms of a WF-net, this is the time it takes to go from source place i to sink place o. One can measure the average lead time over all cases. However, the degree of variance may also be important, i.e., it makes a difference whether all cases take more or less two weeks or if some take just a few hours whereas others take more than one month. The *service level* is the percentage of cases having a lead time lower than some threshold value, e.g., the percentage of cases handled within two weeks.
- The *service time* is the time actually worked on a case. One can measure the service time per activity, e.g., the average time needed to make a decision is 35 minutes, or for the entire case. Note that in case of concurrency the overall service time (i.e., summing up the times spent on the various activities) may be longer than the lead time. However, typically the service time is just a fraction of the lead time (minutes versus weeks).
- The *waiting time* is the time a case is waiting for a resource to become available. This time can be measured per activity or for the case as a whole. An example is the waiting time for a customer who wants to talk to a sales representative. Another example is the time a patient needs to wait before getting a knee operation. Again one may be interested in the average or variance of waiting times. It is also possible to focus on a service level, e.g., the percentage of patients that has a knee operation within three weeks after the initial diagnosis.
- The *synchronization time* is the time an activity is not yet fully enabled and waiting for an external trigger or another parallel branch. Unlike waiting time, the activity is not fully enabled yet, i.e., the case is waiting for synchronization rather than a resource. Consider, for example, a case at marking $[c2, c3]$ in the WF-net shown in Fig. 3.2. Activity e is waiting for *check ticket* to complete. The difference between the arrival time of the token in condition $c4$ and the arrival time of the token in condition $c3$ is the synchronization time.

Performance indicators can also be defined for the *cost dimension*. Different costing models can be used, e.g., Activity Based Costing (ABC), Time-Driven ABC, and Resource Consumption Accounting (RCA) [31]. The costs of executing an activity may be fixed or depend on the type of resource used, its utilization, or the duration of the activity. Resource costs may depend on the utilization of resources. A key performance indicator in most processes is the *average utilization* of resources over a given period, e.g., an operating room in a hospital has been used 85% of the time over the last two months. A detailed discussion of the various costing models is outside of the scope of this book.

The *quality dimension* typically focuses on the “product” or “service” delivered to the customer. Like costs, this can be measured in different ways. One example is customer satisfaction measured through questionnaires. Another example is the average number of complaints per case or the number of product defects.

Whereas verification focuses on the (logical) correctness of the modeled process, performance analysis aims at improving processes with respect to time, cost,

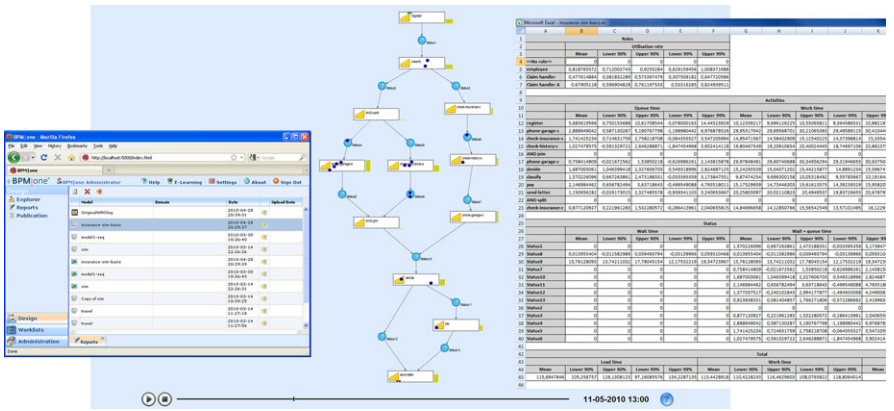


Fig. 3.21 Simulation using BPM|one of Pallas Athena: the modeled process can be animated and all kinds of KPIs of the simulated process are measured and stored in a spreadsheet

or quality. Within the context of operations management many analysis techniques have been developed. Some of these techniques “optimize” the model given a particular performance indicator. For example, integer programming or Markov decision problems can be used to find optimal policies. For the types of process models described in this chapter “what if” analyses using simulation, queueing models, or Markov models are most appropriate. Analytical models typically require many assumptions and can only be used to answer particular questions. Therefore, one needs to resort to *simulation*. Most BPM tools provide simulation capabilities. Figure 3.21 shows a screenshot of BPM|one while simulating a process for handling insurance claims. BPM|one can animate the simulation run and calculate all kinds of KPIs related to time and cost (e.g., lead time, service time, waiting time, utilization, and activity costs).

Although many organizations have tried to use simulation to analyze their business processes at some stage, *few are using simulation in a structured and effective manner*. This may be caused by a lack of training and limitations of existing tools. However, there are also several additional and more fundamental problems. First of all, simulation models tend to *oversimplify* things. In particular the behavior of resources is often modeled in a rather naïve manner. People do not work at constant speeds and need to distribute their attention over multiple processes. This can have dramatic effects on the performance of a process and, therefore, such aspects should not be “abstracted away” [139, 163]. Second, various *artifacts available are not used as input for simulation*. Modern organizations store events in logs and some may have accurate process models stored in their BPM/WFM systems. Also note that in many organizations, the state of the information system accurately reflects the state of the business processes supported by these systems. As discussed in Chap. 1, processes and information systems have become tightly coupled. Nevertheless, such information (i.e., event logs and status data) is rarely used for simulation or a lot of manual work is needed to feed this information into the model. Fortunately, as will

be shown later in this book, process mining can assist in extracting such information and use this to realize performance improvements (see Sect. 9.6). Third, the focus of simulation is mainly on “design” whereas managers would also like to use simulation for *“operational decision making”*, i.e., solving the concrete problem at hand rather than some abstract future problem. Fortunately, *short-term simulation* [139] can provide answers for questions related to “here and now”. The key idea is to start all simulation runs from the current state and focus on the analysis of the transient behavior. This way a “fast forward button” into the future is provided.

3.3.3 Limitations of Model-Based Analysis

Verification and performance analysis heavily rely on the availability of high quality models. When the models and reality have little in common, model-based analysis does not make much sense. For example, the process model can be internally consistent and satisfy all kinds of desirable properties. However, if the model describes an idealized version of reality, this is quite useless as in reality all kinds of deviations may take place. Similar comments hold for simulation models. It may be that the model predicts a significant improvement whereas in reality this is not the case because the model is based on flawed assumptions. All of these problems stem from *a lack of alignment between hand-made models and reality.* Process mining aims to address these problems by establishing a direct connection between the models and actual low-level event data about the process. Moreover, the *discovery techniques discussed in this book allow for viewing the same reality from different angles and at different levels of abstraction*.

Chapter 11
Process Mining Software

The successful application of process mining relies on good tool support. Traditional Business Intelligence (BI) tools are data-centric and focus on rather simplistic forms of analysis. Mainstream data mining and machine learning tools provide more sophisticated forms of analysis, but are also not tailored towards the analysis and improvement of processes. Fortunately, there are dedicated process mining tools able to transform event data into actionable process-related insights. For example, ProM is an open-source process mining tool supporting all of the techniques mentioned in this book. Process discovery, conformance checking, social network analysis, organizational mining, clustering, decision mining, prediction, and recommendation are all supported by ProM plug-ins. However, the usability of the hundreds of available plug-ins varies and the complexity of the tool may be overwhelming for end-users. In recent years, several vendors released dedicated process mining tools (e.g., Celonis, Disco, EDS, Fujitsu, Minit, myInvenio, Perceptive, PPM, QPR, Rialto, and SNP). These tools typically provide less functionality than ProM, but are easier to use while focusing on data extraction, performance analysis and scalability. This chapter provides an overview of available tools and trends.

11.1 Process Mining Not Included!

This book revolves around the analysis of behavior based on event data. Fueled by the growing availability of data (“Big Data”), data science emerged as a new discipline. As discussed in Sect. 1.3, data science approaches tend to be process agnostic. Process mining aims for duality (yin and yang) between data-driven forces and process-centric forces (see Fig. 2.1). The process mining spectrum is broad and, as shown in the previous chapters, extends far beyond process discovery and conformance checking. Process mining connects data science and process science (see Fig. 1.7). Hence, it is inevitable that process mining objectives are overlapping with those of other approaches, methodologies, principles, methods, tools, and paradigms. In Sect. 2.5, we discussed the relation to BPM, BPR, BI, Big Data, data

W. van der Aalst, *Process Mining*, DOI 10.1007/978-3-662-49851-4_11

mining, Lean Six Sigma, etc. We posed questions like: “How does process mining compare to data mining?” (Sect. 2.5.2) and “How does process mining compare to Business Intelligence?” (Sect. 2.5.5). Books on data mining and BI seldom cover process mining techniques. The same holds for data mining and BI software. *Defining process mining as a particular type of machine learning, data mining or BI technique, will not extend the actual capabilities of (machine learning, data mining or BI) tools.* Software packages for machine learning and data mining *cannot* deal with process models (i.e., BPMN, EPC, UML, Petri nets, etc.) and do *not* support tasks like conformance checking. One needs dedicated process mining software for this: It is not included!

In the remainder of this chapter, we describe the capabilities of ProM and various commercial process mining tools. However, before doing so, we briefly discuss the market for BI products.

Forrester defines *Business Intelligence* (BI) in two ways. The broad definition provided by Forrester is “BI is a set of methodologies, processes, architectures, and technologies that transform raw data into meaningful and useful information used to enable more effective strategic, tactical, and operational insights and decision-making” [55]. Forrester also provides a second, more narrow, definition: “BI is a set of methodologies, processes, architectures, and technologies that leverage the output of information management processes for analysis, reporting, performance management, and information delivery” [55].

Some of the most widely used BI products are [56]: *IBM Cognos Business Intelligence* (IBM), *Oracle Business Intelligence* (Oracle), *SAP BusinessObjects* (SAP), *MS SQL Server/Power BI* (Microsoft), *MicroStrategy* (MicroStrategy), *QlikView* (QlikTech), *SAS Business Intelligence* (SAS), *TIBCO Spotfire Analytics* (TIBCO), *Jaspersoft BI Enterprise* (Jaspersoft), and *Pentaho BI Platform* (Pentaho). The typical functionality provided by these products includes:

- *ETL* (Extract, Transform, and Load). All products support the extraction of data from various sources. The extracted data is then transformed into a standard data format (typically a multidimensional table) and loaded into the BI system.
- *Ad-hoc querying.* Users can explore the data in an ad-hoc manner (e.g., drilling down and “slicing and dicing”).
- *Reporting.* All BI products allow for the definition of standard reports. Users without any knowledge of the underlying data structures can simply generate such predefined reports. A report may contain various tables, graphs, and scorecards.
- *Interactive dashboards.* All BI products allow for the definition of dashboards consisting of tabular data and a variety of graphs. These dashboards are interactive, e.g., the user can change, refine, aggregate, and filter the current view using predefined controls.
- *Alert generation.* It is possible to define events and conditions that need to trigger an alert, e.g., when sales drop below a predefined threshold an e-mail is sent to the sales manager.

The mainstream BI products from vendors such as IBM, Oracle, SAP, and Microsoft do *not* support process mining. All of the systems mentioned earlier are

data-centric and are *unaware* of the processes the data refers to. The focus is on fancy-looking dashboards and rather simple reports, instead of a deeper analysis of the data collected. This is surprising as the “I” in BI refers to “intelligence”. Unfortunately, the business ~~un~~intelligence market is dominated by large vendors that focus on monitoring and reporting rather than analytics. Data mining or statistical analysis are often added as an afterthought.

Most BI tools provide interfaces to data mining tools. For example, open-source BI products from organizations like Jaspersoft and Pentaho can connect to open-source data mining tools such as *WEKA* (Waikato Environment for Knowledge Analysis, weka.wikispaces.com), *RapidMiner* (www.rapidminer.com), *KNIME* (Konstanz Information Miner, www.knime.org), and *R* (www.r-project.org). These provide more “intelligence” than mainstream BI tools.

WEKA is a widely-used prototypical example of a data mining tool [190]. WEKA supports classification (e.g., decision tree learning), clustering (e.g., k-means clustering), and association rule learning (e.g., the Apriori algorithm). WEKA expects so-called “arff” files as input. Such a file stores tabular data such as shown in Tables 4.1, 4.2, and 4.3. It is impossible to directly load an event log into WEKA. However, it is possible to convert XES or MXML data into tabular data that can be analyzed by WEKA [42]. After conversion each row either corresponds to an event or a case. For example, it is possible to extract variables like flow time and the frequency of some activity for each case. Similarly, it is possible to create a table where each row lists the attributes of some event. However, either way, the original event notion is lost. This illustrates that data mining tools, like the mainstream BI products, are data-centric rather than process-centric.

Tools such as RapidMiner, KNIME, and R are extendible. For example, RapidMiner provides a marketplace where users can acquire additional building blocks (e.g., for text mining). *RapidProM* (www.rapidprom.org), available through the RapidMiner marketplace, provides a collection of process mining building blocks based on ProM plug-ins (see Sect. 11.3.3). This way users of RapidMiner are able to use process mining techniques without installing a separate process mining tool [23, 97].

In general, one cannot assume that BI and data mining tools provide any process mining capabilities. Fortunately, plenty of dedicated process mining tools are available. These are discussed in the remainder of this chapter.

11.2 Different Types of Process Mining Tools

Before describing concrete process mining tools, we first discuss different ways of characterizing process mining software.

Potentially, there may be very different groups of users interacting with process mining software. On the one hand, there may be experts that need to be able to answer “one of a kind” questions requiring ad-hoc data extractions, complex data transformations, and sophisticated analysis techniques. On the other hand, there can

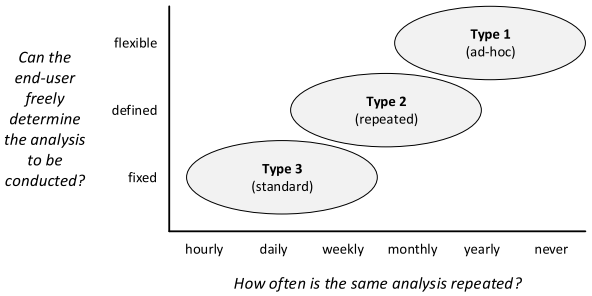


Fig. 11.1 Three types of use cases: *Type 1* (for ad-hoc questions requiring data exploration/extraction and problem-driven selection of analysis techniques), *Type 2* (for repeated questions in a known setting but possibly still requiring configuration), and *Type 3* (for standard questions in a fixed pre-configured setting)

be end-users that just want to look at standard overviews (“process-centric dashboards”) generated using process mining.

The spectrum of process mining use cases can be characterized through the following two questions:

- *How often is the same analysis repeated?*
- *Can the end-user freely determine the analysis to be conducted?*

Fig. 11.1 defines three types of use cases based on answers to these two questions.

Use cases of *Type 1* (ad-hoc) require a spreadsheet-like tool: questions are ad-hoc and the user needs to have complete freedom to perform analysis. The analysis process is iterative and undefined. The results of one analysis step may lead to unanticipated additional data extractions (or transformations) to enable the next analysis step. Analysis workflows are unique and seldom repeated.

Use cases of *Type 2* (repeated) involve questions that are recurring, but at a lower frequency. Analysis workflows may be predefined but not completely fixed. Customization may be needed and the interpretation of the results requires knowledge of process mining and understanding of the data.

Use cases of *Type 3* (standard) involve routine questions that are recurring frequently. The different analysis views are fixed and no customization is possible. The user only needs to understand predefined dashboard-like views.

The three types are on the diagonal in Fig. 11.1. Use cases not on the diagonal do not make much sense. For example, we cannot provide a predefined dashboard for “one of a kind” questions (corresponding to the combination of “never” and “fixed” in Fig. 11.1). Moreover, if there is a continuous need to answer the same question based on the latest data, then there is no need to explore the data in an ad-hoc manner (i.e., also the combination of “hourly” and “flexible” in Fig. 11.1 makes no sense).

Process mining tools may be tailored to one of the three types in Fig. 11.1. For example, a tool like Disco (Fluxicon) is comparable to a spreadsheet program (but for “behavior” rather than “numbers”, see Sect. 1.3). The user can load the data of interest, pick a particular view, and get immediate results without any system configuration. Such style of interaction is good for exploration and fast results (*Type 1*),

but less suitable for end-users that do not understand the underlying data and analysis techniques (*Type 3*).

The initial investment for a *Type 1* analysis is low, but less suitable for situations where many users repeatedly need to do the same type of analysis. The initial investment for a *Type 3* analysis is much higher. An expert needs to configure the way data is extracted and define the views on the data provided to end users. However, after the initial investment, analysis is easier and highly repeatable. *Type 2* analysis is in-between *Type 1* and *Type 3*. Use cases of *Type 2* benefit from analysis workflows that are (partly) predefined but not completely fixed.

Another way to categorize process mining software is based on the way it is bundled:

- *Dedicated* process mining software—pure play process mining tools devoted to the analysis of event data and processes.
- *Embedded* process mining software—tools that provide process mining functionality, but that are embedded in a larger suite.

Most of the tools discussed in this chapter fall in the first category. However, process mining functionality may also be embedded in a larger BPM, ERP, BI or data mining product as an add-on. RapidProM, an extension of RapidMiner, is an example of embedded process mining software [97].

Process mining tools can also be classified based on their “openness”:

- *Open-source* process mining software—the source code is publicly available. Depending on the license other parties can extend, change, or redistribute the software.
- *Closed-source* process mining software—propriety software whose source code is not published and cannot be changed or extended.

The commercial process mining tools described in Sect. 11.4 are closed-source. ProM is an example of an open-source tool.

All process mining tools are able to discover process models, but the types of models learned from event data vary. We distinguish three classes of models:

- *Informal* process models—“boxes and arrows” diagrams not having a formal interpretation that can be related to traces in the event log.
- *Formal low-level* process models—transition systems, Markov chains, episodes, sequences, etc.
- *Formal high-level* process models—end-to-end models allowing for choices, concurrency, loops, etc. This includes BPMN models, EPC models, UML activity diagrams, Petri nets, process trees, etc.

Formal models have executable semantics. Informal models are drawings composed of boxes and arrows without a clear relation to the traces in the event log. Such informal diagrams do not distinguish between choice and concurrency (there are no AND/XOR/OR-gateways/connectors/operators). A model is formal if, given a sequence of events, one can determine whether it fits or not. Process mining tools are characterized by the process models they support. Most commercial process

mining tools use a mixture of informal and low-level models (see Sect. 11.4.2). The fact that a discovered model can be saved in BPMN format (or any other format with AND/XOR/OR-gateways/connectors/operators) does not imply that the model can be interpreted as such.

Process mining starts from event data. Process mining tools may have different mechanisms to get event data:

- *File*. Events are stored in a XES, MXML, Excel, or CSV file.
- *Database*. Events are loaded from a database system, for example via a JDBC connection. Several tools support incremental event loading, i.e., periodically the database is inspected for new data.
- *Adapter*. Events are loaded from a particular application (e.g., SAP, Sharepoint, or SalesForce) through a dedicated piece of software. In most cases events can be loaded incrementally.
- *Streaming*. The process mining tool works on a stream of events emitted through an event bus or web service. Events are captured as they occur and not retrieved from a file, database, or application at a later point in time.

The process mining software may run locally or remotely. The event data typically resides at the same location. We distinguish three types of deployments:

- *Stand-alone*. The software runs locally, e.g., on the laptop used for analysis.
- *On premise*. The back-end of the software does not run locally, but on a server inside the organization.
- *Cloud*. The software runs on a server outside the organization.

Some products offer multiple forms of deployment. This is not only a technological decision, but also related to privacy laws, security, and ethics. For example, the cloud provider may store event data on a server in a different country.

The refined process mining framework (Sect. 10.1) identifies the following activities:

- *Discover*—learning (process) models from event data;
- *Enhance*—repair or extend models (adding additional perspectives to a model, e.g., to show bottlenecks);
- *Diagnose*—model-based process analysis;
- *Detect*—comparing de jure models with current “pre mortem” data (events of running process instances) to detect deviations at runtime;
- *Check*—checking conformance by comparing historic “post mortem” data with de jure models (e.g., to pinpoint deviations and quantify compliance);
- *Compare*—comparing de jure models with de facto models to see whether reality deviates from what was planned or expected;
- *Promote*—transferring “best practices” (learned from event data) to the de jure model;
- *Explore*—exploring business processes at run-time using a combination of event data and models;
- *Predict*—making process-related predictions, e.g., the remaining flow time and the probability of non-compliance;

- *Recommend*—supporting operational processes by recommending suitable actions (e.g. to minimize costs or time).

Process mining software can be characterized by the activities supported. For example, all tools support activity *discover*, but only few support activities like *predict* and *recommend*.

In the remainder, we first describe ProM and then provide an overview of other process mining tools, including 11 commercial products.

11.3 ProM: An Open-Source Process Mining Platform

ProM is the leading open-source process mining tool. The lion’s share of academic research is conducted by using and extending ProM. Moreover, the commercial process mining tools discussed in Sect. 11.4 are based on ideas first developed in the context of ProM. Therefore, this section first introduces ProM which is tailored towards use cases of *Type 1* (see Fig. 11.1).

11.3.1 Historical Context

In 2002, there were several, rather simple, stand-alone process mining tools available. Examples of tools developed around the turn of the century include: *MiMo* (α-miner based on ExSpect), *EMiT* (α-miner taking transactional information into account), *Little Thumb* (predecessor of the heuristic miner), *InWolvE* (miner based on stochastic activity graphs), and *Process Miner* (miner assuming structured models) [156]. At this time, several researchers were building simple prototypes to experiment with process discovery techniques. However, these tools were based on rather naïve assumptions (simple process models and small but complete data sets) and provided hardly any support for real-life process mining projects (scalability, intuitive user interface, etc.). Clearly, it did not make any sense to build a dedicated process mining tool for every newly conceived process discovery technique. This observation triggered the development of the *ProM framework*, a “plug-able” environment for process mining using MXML as input format. The goal of the first version of this framework was to provide a *common basis* for all kinds of process mining techniques, e.g., supporting the loading and filtering of event logs and the visualization of results. This way people developing new process discovery algorithms did not have to worry about extracting, converting, and loading event data. Moreover, for standard model types such as Petri nets, EPCs, and social networks, default visualizations were provided by the framework.

In 2004, the first fully functional version of the ProM framework (*ProM 1.1*) was released. This version contained 29 plug-ins: 6 mining plug-ins (the classic α-miner, the Tshinghua α miner, the genetic miner, the multi-phase miner, the social network miner, and the case data extraction miner), 7 analysis plug-ins (e.g., the

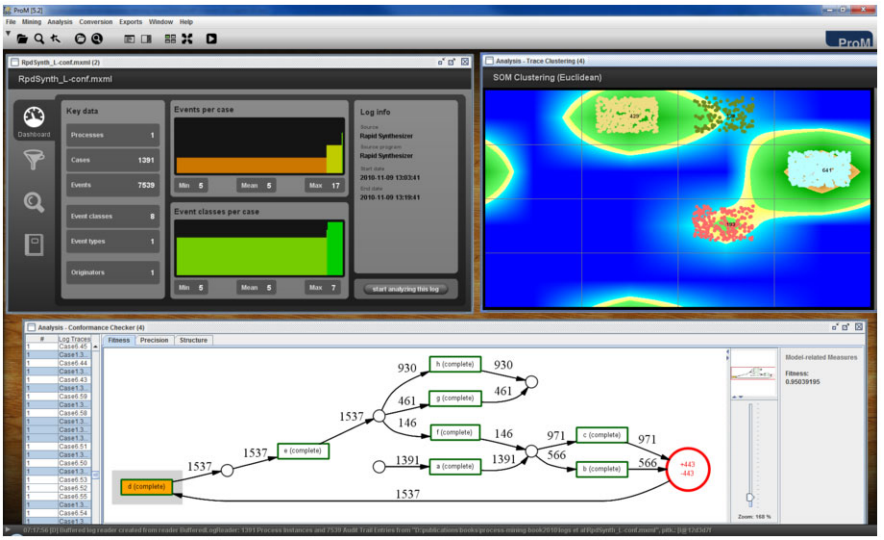


Fig. 11.2 Screenshot of ProM 5.2 showing two of the 286 plug-ins. The *bottom window* shows the *conformance checker* plug-in while checking the fitness of event log L_{full} described in Table 8.1 and WF-net N_2 depicted in Fig. 8.2. The plug-in identifies the conformance problem (the log and model disagree on the position of d) and returns a fitness value computed using the approach presented in Sect. 8.2, $fitness(L_{full}, N_2) = 0.95039195$. The *right window* shows the trace clustering plug-in using Self Organizing Maps (SOM) to find homogeneous groups of cases. The largest cluster contains 641 cases. These are the cases that were rejected without a thorough examination (i.e., traces σ_1, σ_3, σ_{13} in Table 8.1)

LTL checker), 4 import plug-ins (e.g., plug-ins to load Petri nets and EPCs), 9 export plug-ins, and 3 conversion plug-ins (e.g., a plug-in to convert EPCs into Petri nets). Over time more plug-ins were added. For instance, *ProM 4.0* (released in 2006) contained already 142 plug-ins. The 27 mining plug-ins of ProM 4.0 included also the heuristic miner and a region-based miner using Petrify. Moreover, ProM 4.0 contained a first version of the conformance checker described in [121]. *ProM 5.2* was released in 2009. This version contained 286 plug-ins: 47 mining plug-ins, 96 analysis plug-ins, 22 import plug-ins, 45 export plug-ins, 44 conversion plug-ins, and 32 filter plug-ins. Figure 11.2 shows two plug-ins of ProM 5.2. This version already supported most of the process mining techniques presented in this book. For example, the 47 mining plug-ins of ProM 5.2 include most of the discovery algorithms presented in Chap. 7 (genetic mining, heuristic mining, fuzzy mining, etc.). The replay approach presented in Sect. 8.2 was supported by the conformance checker plug-in of ProM 5.2 [121].

The spectacular growth of the number of plug-ins in the period from 2004 to 2009 illustrates that ProM realized its initial goal to provide a platform for the development of new process mining techniques. ProM had become the de facto standard for process mining. Research groups from all over the globe contributed to the development of ProM and people from tens of thousands of organizations down-

loaded ProM (the ProM framework has been downloaded over 130.000 times). In the same period, we applied ProM at numerous organizations, e.g., in the context of joint research projects, Master projects, and consultancy projects. The large number of plug-ins and the many practical applications also revealed some problems. For example, ProM 5.2 can be quite confusing for the inexperienced user who is confronted with almost 300 plug-ins. Moreover, in ProM 5.2 (and earlier versions) the user interface and the underlying analysis techniques are tightly coupled, i.e., most plug-ins require user interaction. It was impossible to embed ProM functionality in data mining tools such as RapidMiner, KNIME, etc. due to this tight coupling.

To be able to run ProM remotely and to embed process mining functionality in other systems, we decided to completely re-implement ProM from scratch. This allowed us to learn from earlier experiences and to develop a completely new architecture based on an improved plug-in infrastructure.

ProM 6 was released in November 2010. This was the first version based on the new architecture and XES rather than MXML. XES, described in Sect. 5.3, is the process mining standard adopted by the IEEE Task Force on Process Mining. Although ProM 5.2 was already able to load enormous event logs, scalability and efficiency were further improved by using OpenXES [64, 65]. Not all plug-ins of ProM 5.2 have been re-implemented in ProM 6. Nevertheless, most of the process mining techniques described in this book are supported by plug-ins developed for ProM 6.

ProM 6 can distribute the execution of plug-ins over multiple computers. This can be used to improve performance (e.g., using grid computing) and to offer ProM as a service. For instance, at TU/e (Eindhoven University of Technology) we use a dedicated process mining grid to handle huge data sets and to conduct large-scale experiments. The user interface has been re-implemented to be able to deal with many plug-ins, logs, and models at the same time. Plug-ins are now distributed over so-called *packages* and can be chained into composite plug-ins. Packages contain related sets of plug-ins. ProM 6 provides a so-called package manager to add, remove, and update packages. Users should only load packages that are relevant for the tasks they want to perform. This way it is possible to avoid overloading the user with irrelevant functionality. Moreover, ProM 6 can be customized for domain-specific or even organization-specific applications.

Figures 11.3 and 11.4 show the selection of the ILP miner plug-in (based on language-based regions, see Sect. 7.4.3) and the resulting process model discovered by ProM 6. ProM 6.5.1a (SilvR+) was released in October 2015. There is also a “ProM Lite” version providing only the most used functionality.

11.3.2 Example ProM Plug-Ins

ProM is open-source software[1] and can be freely downloaded from www.promtools.org or www.processmining.org. Plug-ins can be installed via ProM’s package man-

[1] ProM framework is released under the GNU Lesser General Public License (L-GPL).

Fig. 11.3 Screenshot of ProM 6.5. After loading an event log, a list of applicable plug-ins is shown and the plug-in implementing discovery using language-based regions (ILP miner) is selected

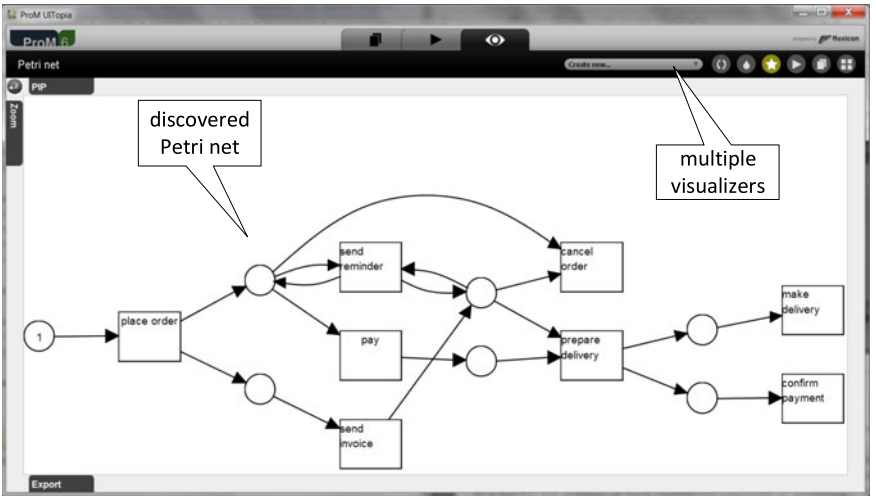


Fig. 11.4 Screenshot of ProM 6.5 showing the Petri net discovered using language-based regions after starting the plug-in selected in Fig. 11.3

ager. Currently, there are over 1500 plug-ins available (including deprecated plug-ins that are no longer supported). The ILP miner plug-in depicted in Fig. 11.4 is just one of these 1500 plug-ins. Hence, it is impossible to provide a complete overview of the functionality of ProM. The reader is encouraged to visit www.processmining.org to learn more about ProM’s functionality and available plug-ins. Here, we only show a few examples.

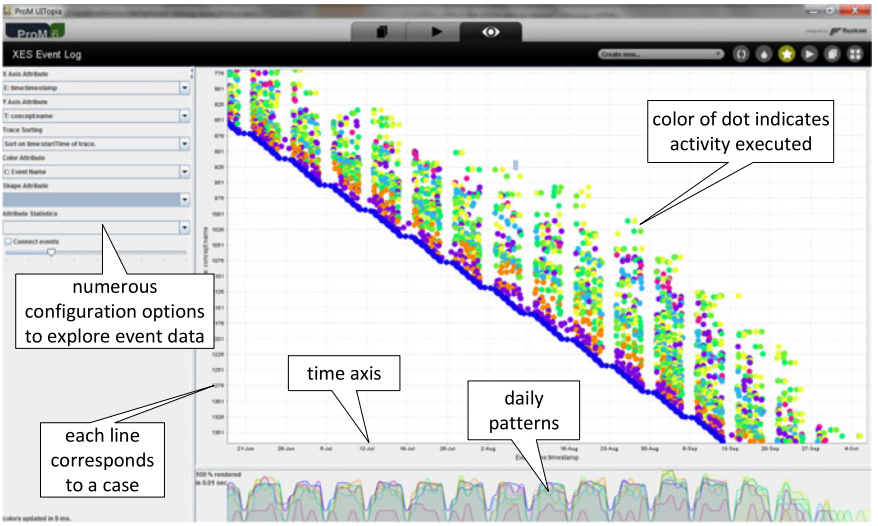


Fig. 11.5 ProM’s dotted chart can be used to explore the event data from different angles

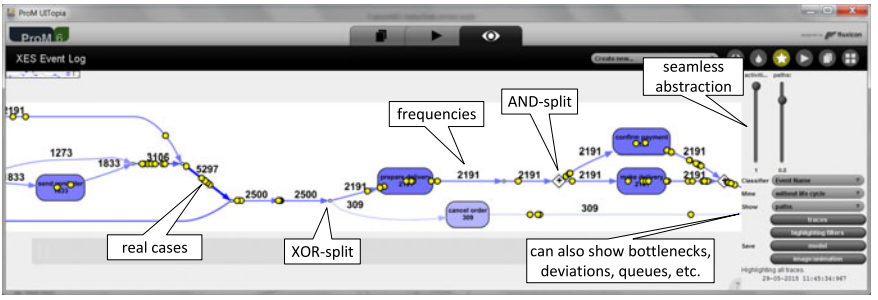


Fig. 11.6 Visual inductive miner replaying the event log on the discovered process model

ProM can load XES, MXML, and CSV files. To extract files from other data sources, tools such as XESame and ProMimport can be used (cf. Sect. 5.3). Figure 11.5 shows a *dotted chart* (see Sect. 9.2). The user can control both axes completely and influence the coloring and shape of the dots.

ProM supports dozens of process discovery algorithms. Next to the ILP miner shown in Fig. 11.4 and the α-algorithm [157], also heuristic mining [183, 184], fuzzy mining [66], genetic process mining [12, 26], and various forms of inductive mining [89–91] are supported. Figure 11.6 shows the *visual inductive miner*. This miner always returns a sound process model and is able to handle large and noisy event logs. Nevertheless, the miner can ensure (if desired) perfects fitness. Results can be converted to Petri nets, EPCs, statecharts and BPMN models. Moreover, the visual inductive miner supports bottleneck analysis and outlier detection.

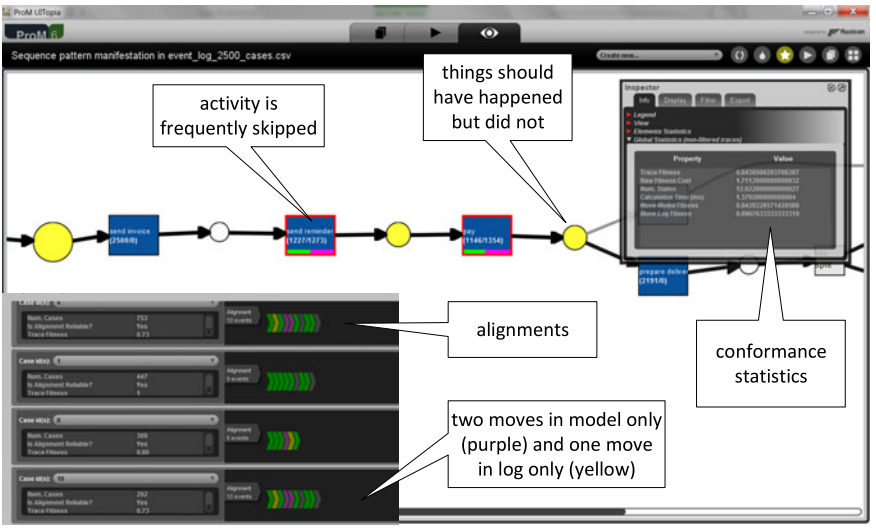


Fig. 11.7 Conformance checking based on alignments (cf. Sect. 8.3)

Conformance checking and performance analysis heavily depend on reliable replay algorithms [169]. Newer plug-ins in ProM rely on alignments as described in Sect. 8.3. Figure 11.7 shows deviations from both the log and model perspective. Next to fitness also notions such as precision are computed [5]. Similar views are provided for performance diagnostics based on alignments.

ProM also supports trace clustering [78], trace alignment [79], and model repair [52]. Next to a variety of procedural models (Petri nets, BPMN, YAWL, EPCs, etc.), ProM also support declarative models. Declare models can be discovered and the conformance of declarative models can be checked.

ProM is not limited to the control-flow and time perspectives. There are plug-ins to create social networks and to discover roles (organizational perspective). There are also plug-ins for decision mining [40, 120]. These plug-ins enhance control-flow models with guards based on the data perspective. Plug-ins can discover so-called *data-aware Petri nets* and check the conformance of such models [40]. Many of these plug-ins create classification problems. See [42] for a ProM plug-in that supports the interaction between process mining and data mining in a generic manner.

The plug-ins mentioned thus far are all related to process mining. However, it should be noted that ProM (both version 5.2 and 6.X) supports process analysis in the broadest sense, e.g., also the analysis techniques mentioned in Sect. 3.3 are supported by ProM or the tools that ProM interfaces with (e.g., CPN Tools). For example, the plug-in “Analyze structural properties of a Petri net” computes transition invariants, place invariants, S-components, T-components, traps, siphons, TP- and PT-handles, etc. The plug-in “Analyze behavioral properties of a Petri net” computes unbounded places, dead transitions, dead markings, home markings, coverability graphs, etc. The “Woflan” plug-in checks the soundness of WF-nets

(cf. Sect. 3.2.3) [179]. Moreover, powerful Petri-net-based analysis tools such as *LoLa*, *Wendy*, *Uma*, and *Petrify* are embedded in ProM as plug-ins.

The hundreds of ProM plug-ins implementing all of the techniques described in this book (and many more) illustrate the applicability and broadness of process mining.

11.3.3 Other Non-commercial Tools

Due to the success of ProM in the academic community, there are only a few other non-commercial process mining tools. Research groups all over the world have contributed to the 1500 plug-ins in ProM (also see the list of organizations mentioned in the Acknowledgements). Next to ProM, most other tools are commercial (cf. Sect. 11.4). A few notable exceptions are described next.

11.3.3.1 PMLAB

PMLAB is a scripting environment for process mining developed by the group of Josep Carmona at Universitat Politècnica de Catalunya in Barcelona. PMLAB can load XES, MXML, and CSV files and different analysis steps can be chained together in scripts. A variety of process discovery techniques based on the theory of regions and satisfiability modulo theories are supported. Tools such as *Genet*, *Petrify*, *Rbminer*, and *Dbminer* can be invoked from PMLAB. These tools are mostly based on state-based regions [34]. As shown in Sect. 7.4, an event log can be converted into a transition system and subsequently synthesized into a Petri net. Classical region theory needs to be extended/relaxed to make it more applicable for process discovery, e.g., Rbminer adapts the classical theory to provide more compact and readable process models [128]. PMLAB uses iPython, a framework for scripting in Python. PMLAB can also invoke ProM plugins through PMLAB scripts. The scripting tool was inspired by MATLAB and Mathematica. However, there are also similarities with RapidMiner, KNIME, and R. PMLAB can be downloaded from https://www.cs.upc.edu/~jcarmona/PMLAB/.

11.3.3.2 CoBeFra

CoBeFra is a benchmarking framework for conformance checking developed at the department of Management Informatics at KU Leuven in Belgium. It is mostly used for the systematic evaluation of process discovery techniques. CoBeFra reads event logs in XES or MXML file format and process models in PNML format. Given an event log and a process model, the tool evaluates the model with respect to dozens of metrics (e.g., various notions of fitness, precision, and simplicity). For large scale experiments (e.g. varying parameters of discovery algorithms to analyze the effects on fitness and precision), the metrics are automatically collected for sets of models and logs. CoBeFra can be downloaded from http://www.processmining.be/cobefra.

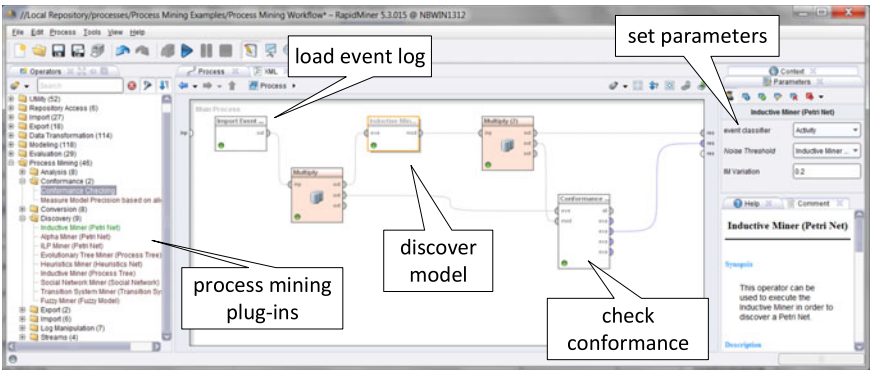


Fig. 11.8 A process mining workflow where an event log is loaded, a model is discovered using the inductive miner, and the result is checked using the conformance checker based on alignments

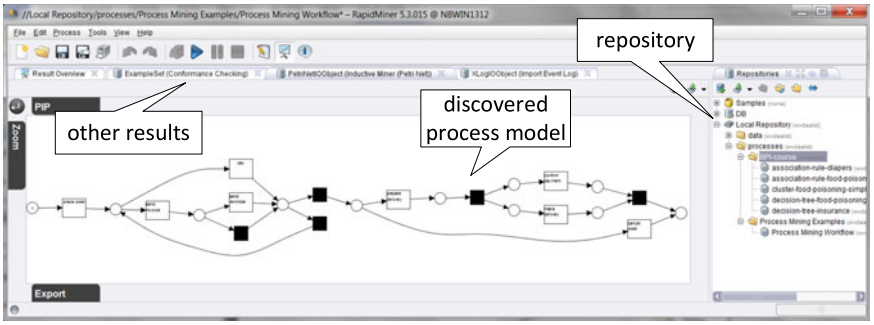


Fig. 11.9 One of the outcomes of the analysis workflow in Fig. 11.8

11.3.3.3 RapidProM

Many tools for data analysis support the definition and execution of *analysis workflows*, sometimes also called *scientific workflows*. For example, widely used tools like RapidMiner, KNIME, and R can chain together building blocks to form such workflows. However, these tools do not support process mining natively. Conversely, ProM does not provide such workflow support. Therefore, *RapidMiner* was extended with process mining plug-ins from ProM. The resulting tool is called *RapidProM* (www.rapidprom.org).

Figure 11.8 shows a process mining workflow created using RapidProM. First, a XES log is loaded. Second, a process model is discovered using the *Inductive Miner—infrequent* (IMf, [89]). The quality of this model is checked using alignments using another building block. The workflow in Fig. 11.8 can be stored and applied to any event log. Figure 11.9 shows one of the output objects produced by the workflow (the discovered process tree was automatically converted to a Petri net).

RapidProM can be used to do large scale experiments [23, 97]. For example, the workflow in Fig. 11.8 can be applied to thousands of event logs without any manual intervention. RapidProM can also be applied to answer recurring questions in a business setting, e.g., to create a report at the end of every week.

RapidProM is available via the *RapidMiner Marketplace*. Many other types of analysis are available in the RapidMiner ecosystem. This facilitates the combination of process mining, text mining, machine learning, data mining, and statistics. For example, cases can be grouped into clusters using standard data mining techniques followed by the application of process mining techniques on each of these clusters.

Whereas ProM is most suitable for use cases of *Type 1*, RapidProM, CoBeFra, and PMLAB are tailored towards use cases of *Type 2* (see Fig. 11.1).

11.4 Commercial Software

Several commercial process mining tools emerged on the market in recent years. Compared to ProM these tools are easier to use, but provide less functionality than the 1500 plug-ins in ProM. This lowers the threshold for using process mining significantly.

This section provides an overview of the commercial process mining tools currently on the market. *The goal is not to give detailed information on specific tools or to provide checklists.* The market and tools change rapidly. Most of the tools described did not exist when the first version of this book was published in 2011 [140]. Moreover, the capabilities of tools change with every release and usability and scalability cannot be expressed in simple checklists. *Hence, organizations that are selecting a commercial process mining tool are urged to evaluate the tools based on concrete questions and datasets.*

After illustrating some of the commercial tools in Sect. 11.4.1, we share a few general insights based on experiences with currently available process mining tools in Sect. 11.4.2.

11.4.1 Available Products

Table 11.1 lists 11 process mining tools in alphabetical order: *Celonis Process Mining* (Celonis), *Disco* (Disco), *Enterprise Discovery Suite* (EDS), *Interstage Business Process Manager Analytics* (Fujitsu), *Minit* (Minit), *myInvenio* (myInvenio), *Perceptive Process Mining* (Perceptive), *QPR ProcessAnalyzer* (QPR), *Rialto Process* (Rialto), *SNP Business Process Analysis* (SNP), and *webMethods Process Performance Manager* (PPM). For tools with a longer name, the shorter name between brackets is used. For example, "webMethods Process Performance Manager" is abbreviated to PPM.

Tools like Disco, Fujitsu, QPR, and PPM have been around for a few years. Minit, myInvenio, and Rialto emerged very recently (in 2015). Tools like *Process*

Table 11.1 Overview of commercial process mining tools

Short name	Full name of tool	Version	Vendor	Webpage
Celonis	Celonis Process Mining	4.0	Celonis GmbH	www.celonis.de
Disco	Disco	1.9	Fluxicon	www.fluxicon.com
EDS	Enterprise Discovery Suite	4	StereoLOGIC Ltd	www.stereologic.com
Fujitsu	Interstage Business Process Manager Analytics	12.2	Fujitsu Ltd	www.fujitsu.com
Minit	Minit	1.0	Gradient ECM	www.minitlabs.com
myInvenio	myInvenio	1.0	Cognitive Technology	www.my-invenio.com
Perceptive	Perceptive Process Mining	2.7	Lexmark	www.lexmark.com
QPR	QPR ProcessAnalyzer	2015.5	QPR	www.qpr.com
Rialto	Rialto Process	1.5	Exeura	www.exeura.eu
SNP	SNP Business Process Analysis	15.27	SNP Schneider-Neureither & Partner AG	www.snp-bpa.com
PPM	webMethods Process Performance Manager	9.9	Software AG	www.softwareag.com

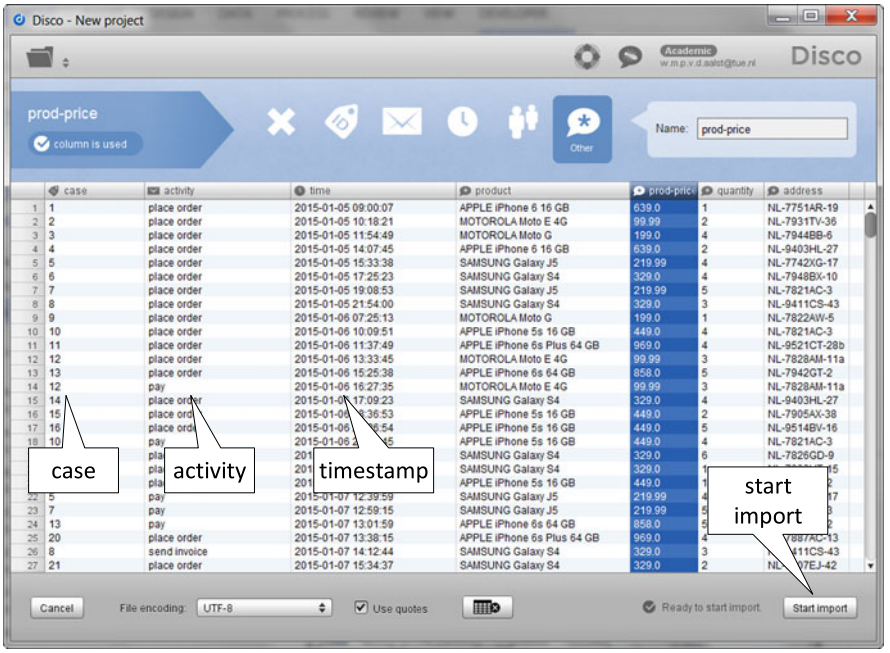


Fig. 11.10 Disco allows for the easy import of CSV files and supports process mining formats such as XES and MXML

Discovery Focus (Iontas/Verint Systems) and *Enterprise Visualization Suite* (Businesscape) are no longer available. Earlier products such as *Reflect|one* by Pallas Athena and *Reflect* by Futura Process Intelligence where further developed as part of the Perceptive suite of BPM tools. It is interesting to note that both Pallas Athena and Futura Process Intelligence were selected as “Cool Vendor” by Gartner in 2009 because of their process mining capabilities. Reflect was the first dedicated commercial process mining tool. The *ARIS Process Performance Manager* (PPM) was initially developed by IDS Scheer. Process mining capabilities were added later and PPM is now part of Software AG’s webMethods Operational Intelligence Platform.

As mentioned, it is not our goal to discuss particular tools in detail. However, we show a few screenshots to provide an impression of typical capabilities of available tools. Figure 11.10 shows a screenshot of Disco while loading a CSV file. The columns can be mapped onto process mining concepts such as case, activity, timestamp, and resource. Disco automatically suggests an initial mapping (including the format to be used for timestamps) that can be adapted. Disco can also load and save event logs in XES and MXML format. Researchers often use Disco for an initial analysis of the data (involving filtering, exploration, and bottlenecks analysis) after which XES files are saved for further analysis using ProM. The discovery algorithm used by Disco can be viewed as an improved and further developed version of ProM’s Fuzzy Miner [66]. The scalability and robustness of Disco are much better than the original Fuzzy Miner. Disco is easy to use and learn, and lowers the barrier

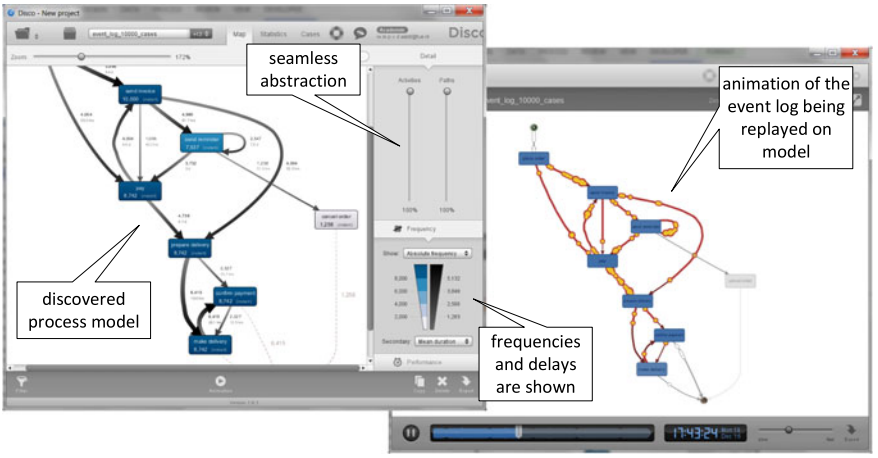


Fig. 11.11 The discovery algorithm of Disco is a further development of the Fuzzy miner and event data can be replayed at the selected abstraction level

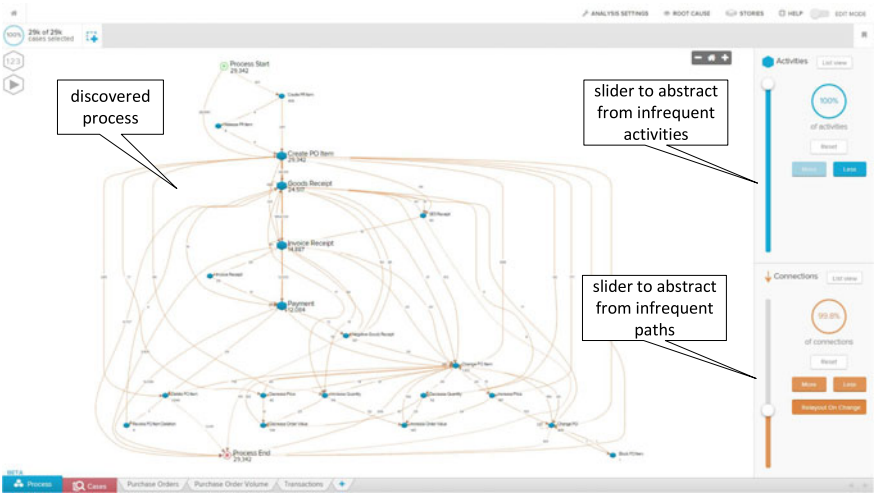


Fig. 11.12 A process model discovered using Celonis showing all activities in the event log

to get started with process mining significantly. Figure 11.11 shows a discovered process model and an animation based on the underlying event data. Animations can be saved as movies and show behavior that changes over time.

Figure 11.12 shows a process model discovered using Celonis. Celonis can load event data from CSV and XES files or database management systems such as SAP HANA, Oracle DB, MSSQL, MYSQL, PostgreSQL and IBM DB2. It is often used in conjunction with SAP. Events are stored in an OLAP-like data structure. Like Disco, Celonis provides sliders to seamlessly simplify models (if desired). In

Fig. 11.13 Animation of the process obtained by replaying the event data on a simplified process model and three charts showing trends in the event data

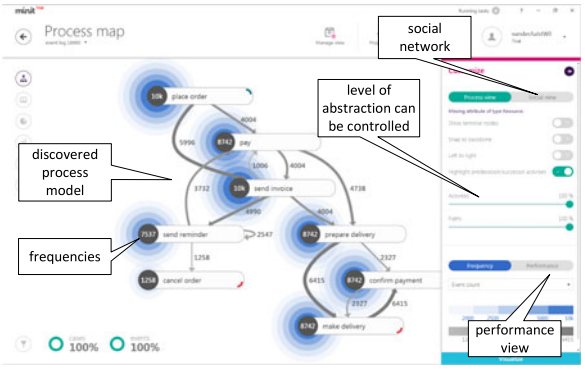


Fig. 11.14 A process model discovered using Minit

Fig. 11.13, a simplified model is used to show an animation of the process. Process related information can also be summarized in column-, line-, area-, pie-charts or tables. This is illustrated by the three charts in Fig. 11.13.

Figure 11.14 shows a screenshot of a model discovered using Minit. Minit also supports XES and uses a discovery algorithm similar to ProM’s Fuzzy Miner. Like Disco and Celonis, Minit is able to handle large event logs efficiently.

Like Minit, myInvenio became available in 2015. These tools illustrate the growing interest in process mining. Figure 11.15 shows a process model discovered using myInvenio. Conformance checking is supported by comparing a reference model (e.g., specified in BPMN or XPDL) with the discovered process model. The differences can be highlighted as shown Fig. 11.15.

Figure 11.16 shows a process model discovered using Perceptive Process Mining. Performance-related information is mapped onto the process model (durations and

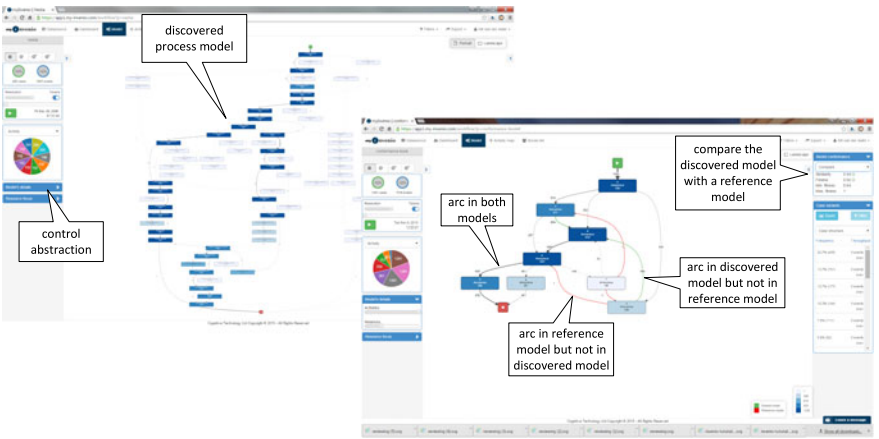


Fig. 11.15 A process model discovered using myInvenio (*left*) and the comparison of a reference model and a discovered model (*right*)

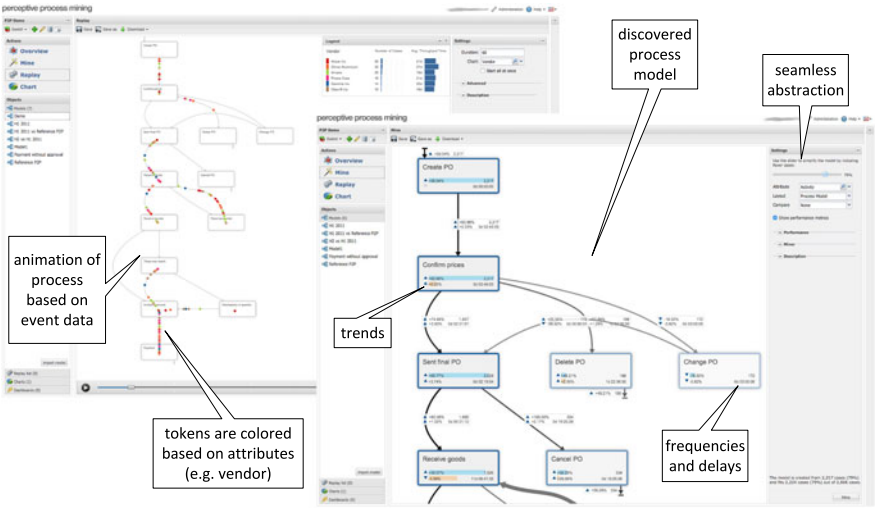


Fig. 11.16 The process model discovered using Perceptive is used to signal trends in performance (*right*) and to animate the process using “colored tokens”(*left*)

frequencies). Perceptive also shows trends, e.g., bottlenecks that are growing over time. The animation in Fig. 11.16 uses colored tokens. The coloring can be based on any case attribute (e.g., the vendor). This helps to spot differences between distinct groups of cases.

Figure 11.17 shows screenshots of four other process mining tools. Each process shown was discovered using event data.

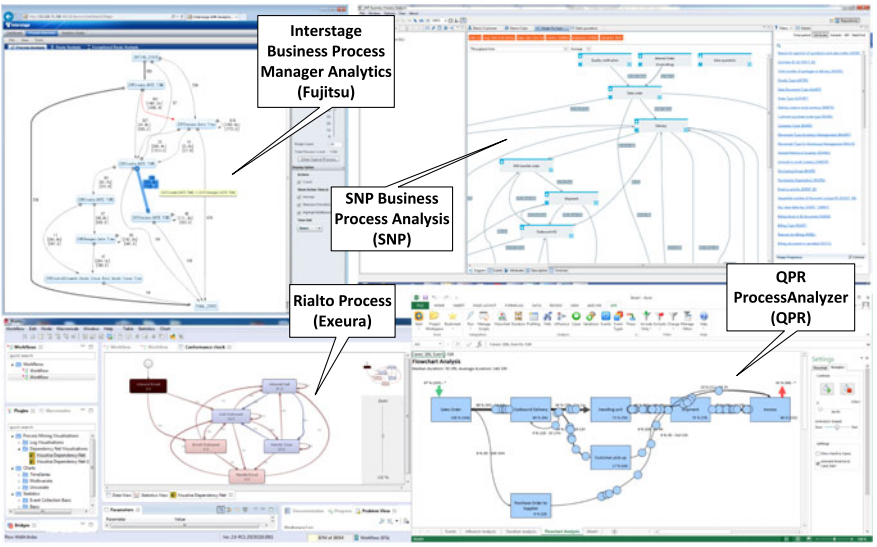


Fig. 11.17 Screenshots of four additional process mining tools: Fujitsu Interstage Business Process Manager Analytics (Fujitsu), SNP Business Process Analysis (SNP), QPR ProcessAnalyzer (QPR), and Exeura Rialto Process (Rialto)

11.4.2 Strengths and Weaknesses

The screenshots in Figs. 11.10–11.17 show that process mining capabilities are readily available in commercial tools. None of the products covers the range of process mining capabilities supported by the hundreds of available ProM plug-ins. However, ProM requires process mining expertise and is not supported by a commercial organization. Hence, it has the advantages and disadvantages common for open-source software. Fortunately, the 11 process mining tools listed in Table 11.1 nicely complement ProM.

On the one hand, there are quite some commonalities among the commercial tools (as illustrated by the screenshots in Figs. 11.10–11.17). On the other hand, there are major differences in usability and scalability. Some focus more on use cases of *Type 1* (e.g., Disco, Minit, and myInvenio) whereas other tools focus more on use cases of *Type 3* (e.g., Celonis and PPM). Organizations selecting a commercial process mining product are urged to do a pilot project where a few products are applied to organization-specific data and questions.

Despite the differences between the tools, we can make some general observations.

11.4.2.1 Limited Support for Concurrency

If a group of activities is not always executed in the same order, we would like to avoid the situation where all activities are connected to one another. Yet, process

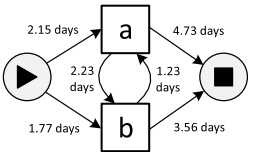


Fig. 11.18 Most tools that do not allow for concurrency have difficulties handling event logs like $L_{par} = [\langle a, b\rangle^{100}, \langle b, a\rangle^{100}]$: Due to the introduction of loops, the model allows for non-observed traces like $\langle a, b, a\rangle$, $\langle a\rangle$, $\langle b, a, b, a, b, a, b\rangle$, $\langle b\rangle$, etc.

discovery techniques that do not support concurrency do exactly that. In Sect. 11.2, these purely sequential models were called low-level models. The models discovered by such techniques tend to be very Spaghetti-like. Moreover, sequential models where every activity appears only once tend to be severely underfitting (e.g., parallel activities are turned into loops).

To illustrate the problem consider the artificial event log $L_{par} = [\langle a, b\rangle^{100}, \langle b, a\rangle^{100}]$. Clearly, there is no loop and one would expect that the discovered model shows that a and b both happen once per case. However, if a and b cannot be concurrent and the tool has one node per activity, then the tool is forced to introduce loops allowing for traces like $\langle a, b, a, b, a\rangle$ (see Fig. 11.18). Clearly, this model is underfitting and not adequately reflecting the observed behavior.

Some of the commercial tools do not support concurrency at all (e.g., SNP). Perceptive Process Mining offers two mining algorithms: a genetic algorithm able to discover concurrency (but time-consuming and not scalable) and a simpler, better performing, algorithm based on the directly follows relation having the problem mentioned above.

Also Disco deals with concurrency different from the algorithms described in Chaps. 6 and 7. Parallelism in Disco is discovered only if two activity instances for the same case overlap. This implies that concurrency cannot be discovered in event logs without explicit transactional information (e.g., when there are just complete events). If activities are interleaved (i.e., not overlapping), then the arrows are suppressed, unless the slider is moved up to ensure perfect fitness. Using the terminology introduced in Sect. 11.2, Disco shifts from an informal model with concurrency to a formal low-level model without concurrency to ensure correctness.

Other tools have similar issues and are often less clear about this. They operate in the space between informal models and low-level models, thus making interpretation tricky. Consider the delays in Fig. 11.18. When does the process end—4.73 days after the (last) completion of activity a or $2.23 + 3.56 = 5.79$ days after the (last) completion of activity a? Probably none of the two answers is right, thus illustrating the confusion.

To summarize—*None of the commercial tools handles concurrency adequately.* There are at least two reasons for this. First of all, simple algorithms are used to ensure scalability and transparency. Second, the models learned have informal semantics. The latter is interesting because several tools claim to support BPMN and can export models to BPM systems. This may be lead to misleading results. Most

tools do not show explicit AND/XOR-splits/joins. Adding logic when saving models will lead to confusion and may result in models that are not sound (e.g., having deadlocks).

As long as process models are interpreted as "pictures" this is not a problem. However, the way that models need to be interpreted *also influences frequencies and performance results*. For example, if it is unclear whether things need to be synchronized or not, computed waiting times cannot be trusted. The inductive mining techniques presented in Sect. 7.5 show that it is possible to discover concurrency without creating unsound or imprecise models. The different inductive mining algorithms (IM, IMF, IMC, IMD, IMFD, IMCD, etc.) always produce sound models and are highly scalable. Some of these algorithms even provide formal guarantees (e.g., perfect fitness).

11.4.2.2 Limited Support for Conformance Checking

Informal models that can only be interpreted as "pictures" cannot be used for conformance checking. Currently, there is no commercial tool that computes alignments or that is able to apply some other replay algorithm to precisely diagnose deviations in the presence of concurrency. The reasons are the same as before: scalability (computing alignments may be too time consuming) and informal semantics (e.g., not being able to distinguish between AND-joins and XOR-joins).

Conformance checking is not handled by replaying the event log on a precise end-to-end process model. Instead one or more of the following approaches are used:

- *Rule based*. The user can specify rules for filtering. For example, Disco and Celonis can be used to define a wide variety of rules (e.g., "a is followed by b and not c and should be executed by a resource not involved in d"). By applying such rules, the event log can be split into conforming and non-conforming cases.
- *Outlier based*. Infrequent paths that deviate from mainstream behavior are manually inspected. By classifying certain paths as deviating, the corresponding cases are tagged as non-conforming.
- *Side-by-side*. The discovered process model and the normative reference model are depicted next to each other. Users need to visually compare models to see deviations.
- *Overlay*. The discovered model and the reference model are stacked on top of each other and differences are highlighted. Figure 11.15 illustrates that myInvenio supports this type of comparison.

Comparing a discovered model and a reference model may lead to incorrect conclusions. Note that the discovered model generalizes over the data, i.e., paths possible in the model may never have happened. This may trigger the detection of deviations that never occurred in reality. The discovered model may also abstract from infrequent behavior. Therefore, rare (but possibly harmful) deviations may remain undetected. However, such peculiar deviations tend to be highly relevant for conformance checking.

Assume that the informal model in Fig. 11.18 was discovered for L_{par} and subsequently compared with a normative model putting a and b in parallel. A visual comparison of the two models would suggest non-existing deviations.

The techniques in Chap. 8 and the plug-ins of ProM show that conformance checking is possible. However, conformance checking can only be supported if the informal models are replaced by formal models (e.g., process trees or Petri nets with a defined initial and final marking). *As long as this functionality is not present, users are forced to capture the real semantics of the normative model in terms of a collection of rules used for filtering.*

11.4.2.3 Performance Perspective is Well Supported

The primary focus of commercial process mining tools is on performance. Each of the tools can visualize bottlenecks in the process. Tools such as Celonis, Perceptive, QPR, Minit, PPM, etc. provide a range of charts. Most of the commercial tools make it possible to quickly find bottlenecks, unnecessary rework and delays.

Note that the problems mentioned earlier may endanger the correctness of performance results. If misalignments and concurrency are not handled well, then the reported results may be incorrect. For example, tools may report negative waiting times if events are reversed or excessive times if events are missing.

11.4.2.4 Data Perspective Not in Models

None of the commercial process mining tools is able to discover data-aware process models. For example, it is impossible to learn guards or perform any other form of decision mining as described in Chap. 9. Conformance checking of models with data is also not supported.

Additional data in the event log can be used in rules for filtering. Moreover, some tools can show the distribution of values for particular groups of cases. However, data are not explicitly related to the process model.

11.4.2.5 Organizational Perspective

Most tools are able to construct a social network (see Chap. 9). It is typically also possible to see the utilization of resources. Nearly all tools consider resource information, roles, and other organizational entities as plain data elements. Hence, the organizational perspective can be handled in the same way as data (e.g., using filtering). Separation of duties (4-eyes principle) can be checked in this way. myInvenio also creates an activity map (a simplified RACI matrix). Normally, a RACI matrix shows the people Responsible, Accountable, Consulted, and Informed for each activity. Event logs need to be enriched to provide this type of information. Sophisticated analysis techniques to optimize work distribution and social network analysis are still missing.

11.4.2.6 Growing Support for XES

Next to tools like ProM, RapidProM, PMLAB, and CoBeFra, the XES standard is supported by a growing number of commercial tools. Currently, Disco, Celonis, Minit, Rialto, and SNP support XES. QPR and myInvenio have announced XES support for the next release. Perceptive, PPM, EDS, and Fujitsu do not (yet) support XES.

XES makes it easier to combine different tools, e.g., using a commercial tool in conjunction with ProM, RapidProM, PMLAB, or CoBeFra.

11.4.2.7 Getting Event Data from Other Sources

Vendors of commercial tools realize that substantial time is spent on extracting data from information systems. In Sect. 11.2, we listed four mechanisms to get event data: file, database, adapter, and streaming. Next to file-based imports of XES, MXML, and CSV, most tools support the extraction of data from JDBC databases. Events can often be loaded from systems such as MySQL, IBM DB2, Oracle DB, SQL Server, PostgreSQL, and SAP HANA.

Often datasets can also be *incrementally* updated (importing only changes since the last import). For example, Disco can retrieve data from a server with the so-called Airlift interface. On the server side of the Airlift connection, arbitrary databases and production systems can be connected.

Systems like Celonis provide additional support to obtain data from SAP systems. Due to the partnership between SAP and Celonis, integration with SAP products like SAP HANA is safeguarded. In fact, most process mining tools support application specific adapters, but the range of systems covered and the quality of these adapters varies per tool.

11.4.2.8 Filtering

Filtering plays a crucial role in most commercial systems. Figure 11.19 shows six types of filtering supported by Disco. Filters can be used to remove individual events or complete cases. For example, one can remove all slow cases, all exceptional cases, etc. One can also specify LTL or Declare-like rules, e.g., activity a should be eventually followed by b (the response constraint in Declare and "$\Box(a \Rightarrow (\Diamond b))$" in LTL). Filtering can be used for ad-hoc conformance checking and plays an important role in root-cause analysis.

Filtering is related to OLAP (see Sect. 12.4). The dimensions in an OLAP cube also split the data based on different criteria. Process mining tools like Celonis store events in multidimensional cubes to facilitate the selection and comparison of particular groups of cases.

11.4.2.9 No Automatic Clustering

Filtering and the selection of dimensions in an OLAP cube are based on user-defined criteria. However, one may also use clustering techniques that automatically group

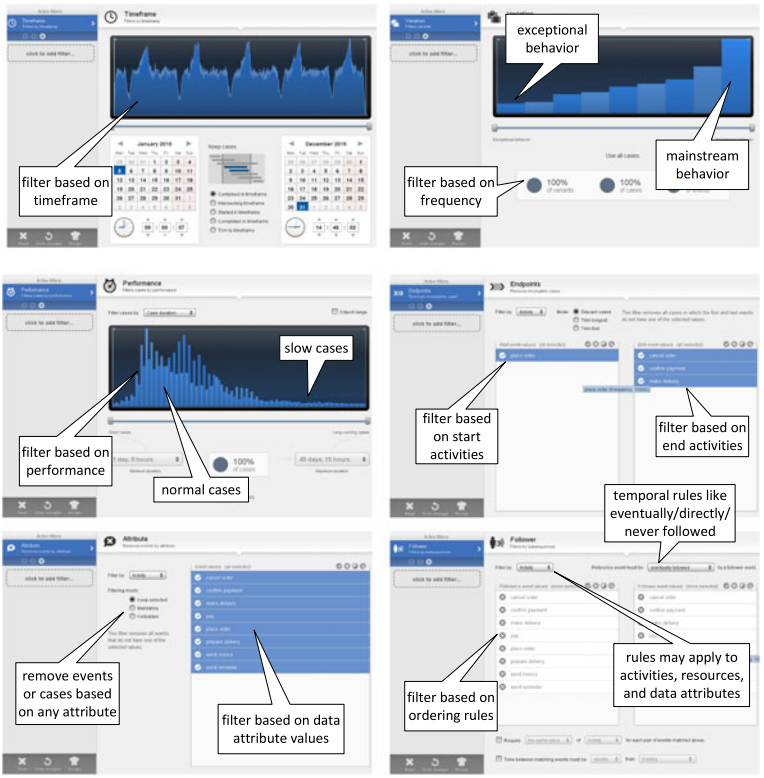


Fig. 11.19 Illustration of the extensive filtering capabilities in commercial systems like Disco

cases that are similar. ProM provides several ways of clustering similar cases based on selected features.

Standard techniques like k-means clustering (see Sect. 4.3 and Chap. 9) can be used as a preprocessing step for process mining [13, 62, 78]. The clusters themselves may already provide novel insights. Moreover, the clusters can often be used to discover multiple simple process models instead of one complex process model.

Surprisingly, clustering is not supported by the current generation of commercial process mining tools.

11.4.2.10 Reporting and Animation

Process mining results need to be communicated. Most tools provide means to create reports, for example, by storing artifacts such as charts, tables, and models. Compared to BI tools, the reporting facilities of process mining tools are often limited.

Disco, Celonis, Perceptive, Minit, QPR, and myInvenio support data-driven process animations. Figures 11.6, 11.11, 11.13, and 11.16 show screenshots where

event logs are animated by replaying them on discovered models. Such animations are instrumental when convincing management. Animation is also a means to support change management: It can be used to create a sense of urgency and to build consensus on root causes.

11.4.2.11 Links to Other Tools

Some of the process mining tools are part of a bigger suite. For example, PPM is part of the webMethods suite and the ARIS family of tools. Perceptive Process Mining was developed as part of Perceptive's BPM suite. It is expected that in time most BPM systems will provide a process mining component (similar to the simulation components in today's BPM systems).

Most process mining tools are able to export process models in a format that can be read by other tools (e.g., BPMN or XPDL). This way the results from process mining can be used as a starting point for modeling, simulation, and documentation.

As mentioned in the context of RapidProM, the interplay between process mining and data mining is extremely valuable. Hence, some process mining tools can export data in a form that can be analyzed by standard data mining tools. Other crossovers of tools are possible. For example, loading process mining results into Excel to create a chart or to compute some statistic.

11.4.2.12 Operational Support

Disco, Celonis, Perceptive, QPR, PPM, and Fujitsu can upload data periodically or incrementally. Analysis views are "refreshed" based on the new data. However, true predictive analyses, as described in Chap. 10, are seldom supported. QPR, PPM, and Rialto report integration efforts with dedicated prediction tools. However, these approaches do not seem to be process specific (i.e., the discovered model is not leveraged for prediction).

11.4.2.13 Scalability

Most of the commercial process mining tools have a good performance in terms of scalability and responsiveness. Some tools can even *handle event logs with billions of events, millions of cases, and hundreds of activities*. Loading such event logs may be time consuming (say up to an hour), but once the log is loaded analysis can be done within a few seconds.

Scalability depends on many different factors and not only the size of the event log. Some types of analysis are sensitive to the average trace length of cases, the number of distinct activities, or the number of attributes per event. Section 12.1.3 describes the key characteristics of logs relevant for scalability.

Organizations selecting a process mining tool are advised to test the scalability of tools on their own data using standard hardware. This is the only way to compare performance in a meaningful way (be aware of indexing and special hardware). See Chap. 12 for techniques to handle even larger event logs.

11.5 Outlook

It is impossible to give a complete overview of all products supporting process mining. Just ProM, the leading open-source process mining framework, already provides more than 1500 plug-ins. These plug-ins cover a wide range of analysis techniques. For example, all process discovery approaches described in this book are supported through ProM plug-ins. Moreover, ProM is not limited to process discovery and also supports conformance checking, social network analysis, bottleneck analysis, decision mining, operational support, verification, model conversion, etc. Most ProM plug-ins aim at use cases of *Type 1*. RapidProM, CoBeFra, and PMLAB support use cases of *Type 2*.

The 11 commercial process mining tools described in this chapter help to lower the threshold for process mining. Next to use cases of *Type 1*, also use cases of *Type 3* are supported using pre-configured dashboards and automated data extraction. Each of the eleven tools aims at supporting less experienced users. Sometimes process mining capabilities are embedded in larger software products. The scalability and usability of most commercial systems is good. Several tools can handle event logs with billions of events. However, compared to ProM there are also typical weaknesses such as the inability to discover concurrency well and the limited support for conformance checking. The focus is on performance analysis rather than conformance checking and precise models.

Since the process mining market is developing fast, readers are advised to test tools using their own event data. Even when tools look similar, differences in terms of practical usability and scalability may be significant.