HAW HAMBURG

Bachelorarbeit

Nico Gnoss

Entwurf und prototypische Umsetzung einer Bibliothek zur generischen Anwendung von Process-Mining

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Nico Gnoss

Entwurf und prototypische Umsetzung einer Bibliothek zur generischen Anwendung von Process-Mining

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Wirtschaftsinformatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Ulrike Steffens
Zweitgutachter: Prof. Dr. Sebastian Rohjans

Eingereicht am: 27. Juni 2019

Nico Gnoss

Thema der Arbeit

Entwurf und prototypische Umsetzung einer Bibliothek zur generischen Anwendung von Process-Mining

Stichworte

Process-Mining, Data Science, Petri-Netz, XES, PNML

Kurzzusammenfassung

Process-Mining ist eine Technik des Prozessmanagements, die es ermöglicht, Businessprozesse auf Basis digitaler Spuren in IT-Systemen, wie z.B. Logfiles aus ERP-Systemen, zu rekonstruieren und zu analysieren. Dabei werden die einzelnen Schritte des Prozesses zusammengefügt und der Prozess in seiner Gesamtheit, z.B. in Form von Petri-Netzen oder BPMN-Modellen, visualisiert. Die Motivation hinter Process-Mining fundiert auf dem Nicht-Vorhandensein von formalen Prozessbeschreibungen oder der Verbesserung von bestehenden Prozessen mit geringer Qualität. Ziel dieser Thesis ist es einerseits den aktuellen State of the Art von Process-Minining darzulegen und andererseits eine Bibliothek zur generischen Anwendung von Process-Mining zu entwickeln.

Nico Gnoss

Title of Thesis

Design and prototype implementation of a library for generic application of process mining

Keywords

Process-Mining, Data Science, PetriNet, XES, PNML

Abstract

Process mining is a technique of process management, which enables business processes to be reconstructed and analyzed based on digital tracks in IT systems, such as e.g. Log

files from ERP-systems. In doing so, the individual steps of the process are put together and the process in its entirety is visualized, e.g. in the form of Petri-Nets or BPMN-models. The motivation behind process mining is based on the lack of formal process descriptions or the improvement of existing low-quality processes. The aim of this thesis is on the one hand to present the current state of the art of process mining and on the other hand to develope a library for generic application of process mining.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Wandel der Gesellschaft

Seit der Jahrtausendwende hat sich die Gesellschaft von überwiegend *analog* zu überwiegend *digital* verschoben (vgl. Hilbert und Lopez 2011). Diese Veränderung hat einen großen Einfluss darauf, wie Menschen miteinander kommunizieren oder wie Unternehmen mit Menschen oder anderen Unternehmen Geschäfte abwickeln (vgl. Manyika u.a. 2011).

Eine der Möglichkeiten der digitalen Gesellschaft ist es, jederzeit, an jedem Ort, beliebige Daten zu sammeln. Die ständige Verbesserung der Micro-Chip-Technologie und der immer günstiger werdende Speicherplatz sorgen dafür, dass große Mengen an Daten gespeichert werden, welche überwiegend in unstrukturierter Form vorliegen. Eine der größten Herausforderungen für Unternehmen heutzutage ist es, Informationen aus den im Informationssystem gespeicherten Daten zu extrahieren, welche in Gewinn umgewandelt werden können (vgl. Aalst 2016, S. 3-4).

Die Wichtigkeit von Informationssystemen spiegelt sich nicht nur durch das rapide Wachstum von Datenmengen wider, sondern auch dadurch, dass datenverarbeitende Systeme eine wichtige Rolle in den heutigen Business-Prozessen spielen, da die digitale und physische Welt immer mehr miteinander verwoben sind.

Ein Beispiel hierfür wäre eine Bank. Im Vergleich zum letzten Jahrhundert, wo Geld überwiegend ein physischer Gegenstand in Form einer Münze oder eines Scheines war, ist es heutzutage nur noch eine digitale Entität. Das Vermögen ist de facto nur eine Zahl, die in einem Informationssystem abgespeichert ist. Kauft man nun ein Produkt bei einem Online-Versandhaus und wählt als Bezahlungsmethode beispielsweise Kreditkarte oder PayPal aus, wird im Endeffekt die eine Zahl, die das Vermögen des Käufers darstellt, verringert und eine andere Zahl, die das Vermögen des Verkäufers darstellt, erhöht; es erfolgt kein Austausch von physischem Geld.

Events

Das unglaubliche Wachstum des digitalen Universums, zusammengefasst unter dem Begriff *Big Data*, macht es möglich sogenannte Events zu speichern und zu analysieren. Events können innerhalb einer Maschine (z.B. Röntgen-Maschine oder Bankautomat), eines Informationssystems (z.B. Bestellung eines Kunden), innerhalb eines Krankenhauses (z.B. Analyse einer Blutprobe) oder eines sozialen Netzwerkes (z.B. Austausch von Nachrichten) stattfinden (vgl. Aalst 2016, S.5).

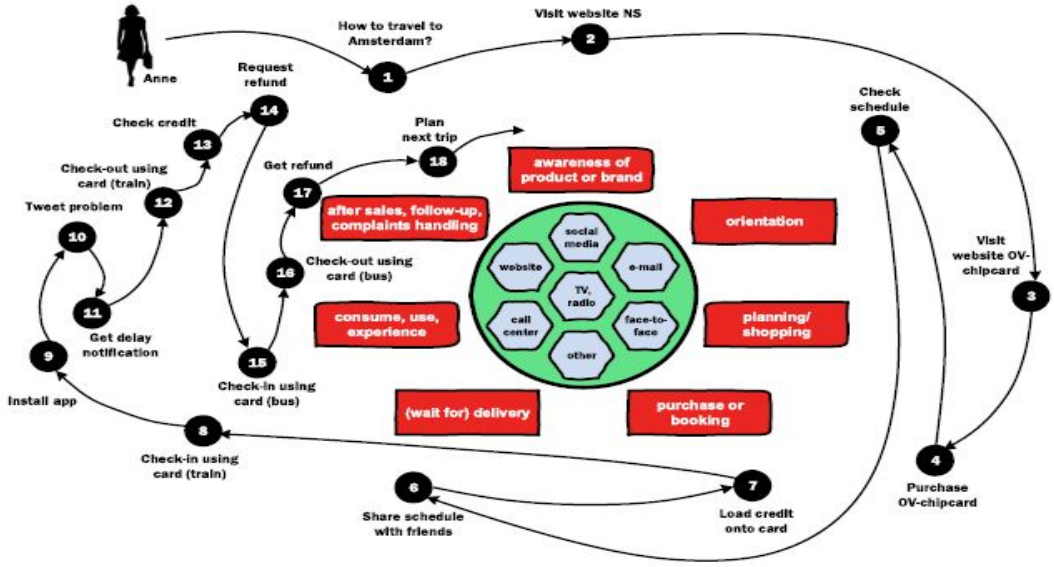

Abbildung 1.1: Entstandene Events bei einer Reise (Aalst 2016, S.7)

Ein Beispiel für den Ablauf von mehreren Events, welches von van der Aalst stammt, könnte eine Reise vom Hauptbahnhof Eindhoven zum Van-Gogh-Museum in Amsterdam sein. Eine Frau hat den Wunsch nach Amsterdam zu reisen und findet mit Hilfe des Internets heraus, dass sie eine bestimmte Chipkarte dafür benötigt. Nachdem sie die Karte gekauft hat, sich den Fahrplan herausgesucht und diesen mit ihren Freunden geteilt hat, lädt sie Guthaben auf diese Karte. Schließlich benutzt sie diese Karte, um mit Bus und Bahn zu fahren, wobei manche Unannehmlichkeiten auftreten, die sie über Twitter teilt und anschließend eine Rückerstattung anfordert. Der detaillierte Ablauf ist in Abbildung 1.1 zu sehen.

Data-Science

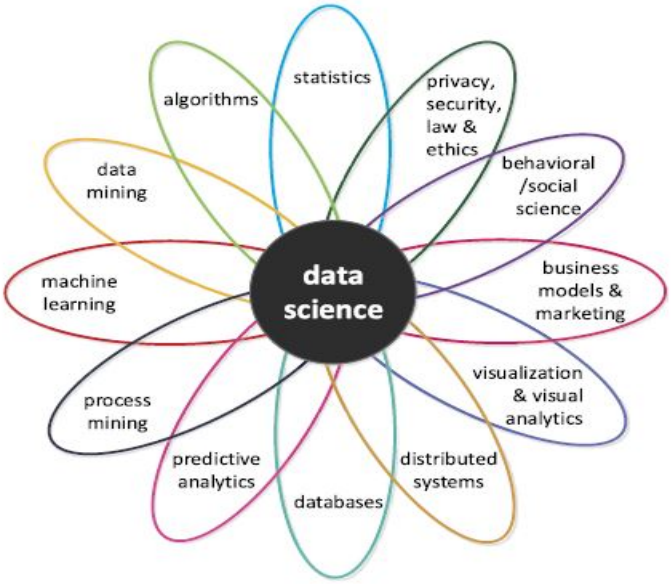

Abbildung 1.2: Bestandteile Data-Science (Aalst 2016, S.12)

Die Begebenheit, dass immer mehr Daten erfasst und gespeichert werden konnten, führte dazu, dass Data-Science als eine neue Wissenschaft entstand:

"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"(Aalst 2016, S.10).

Die Fragen, die mit Hilfe von Data-Science beantwortet werden sollen, können in vier verschiedene Kategorien unterteilt werden (vgl. Aalst 2014, S.14):

- (Reporting) - Was ist passiert?
- (Diagnose) - Wieso ist es passiert?

- (Vorhersage) - Was wird passieren?
- (Empfehlung) - Was sollte passieren?

Data-Science ist eine Vereinigung von verschiedenen sich teilweise überschneidenden Unterdisziplinen, welche in Abbildung 1.2 zu sehen sind. Neben Unterdisziplinen wie z.B. Statistik, Algorithmen, Visualisierung & Visual Analytics, Data-Mining oder Machine-Learning, ist auch Process-Mining zu sehen. Process-Mining fügt dem Data-Mining und Machine-Learning eine Prozessperspektive hinzu. Die untersuchten Daten werden in Verbindung mit expliziten Prozessmodellen gebracht, wie z.B. Petri-Netze oder BPMN-Modellen (vgl. Aalst 2016, S.13).

Process-Science

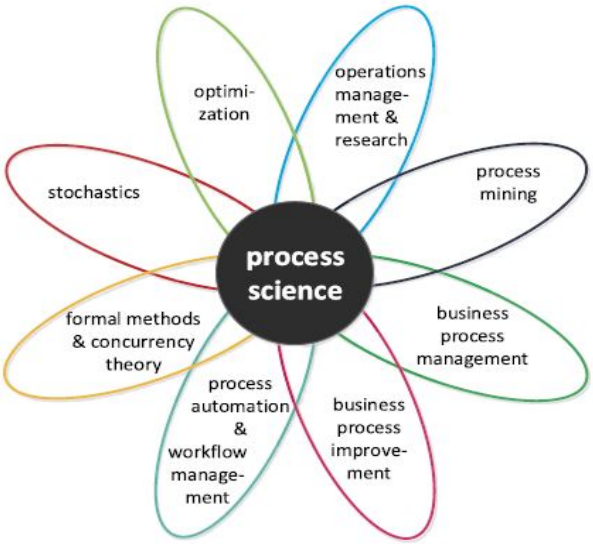

Abbildung 1.3: Bestandteile Process-Science (Aalst 2016, S.16)

In Abbildung 1.2 wurde Process-Mining als Bestandteil von Data-Science beschrieben, was aber keine allgemeine Auffassung ist. Data-Science wird als Wissenschaft angesehen, welche entstanden ist, um sogenannte datengetriebene Probleme und Fragestellungen zu lösen. Die Wissenschaft, welche sich mit prozessgetriebenen Problemen und Fragestellungen beschäftigt, wird als Process-Science bezeichnet. Process-Science kombiniert Wissen

aus der Informationstechnologie und Managementwissenschaft, um operationale Prozesse zu verbessern (vgl. Aalst 2016, S.15).

Neben Optimierung, Business-Process-Management oder Workflow-Management ist in Abbildung 1.3 Process-Mining als einer der Unterdisziplinen von Process-Science zu erkennen.

Process-Mining

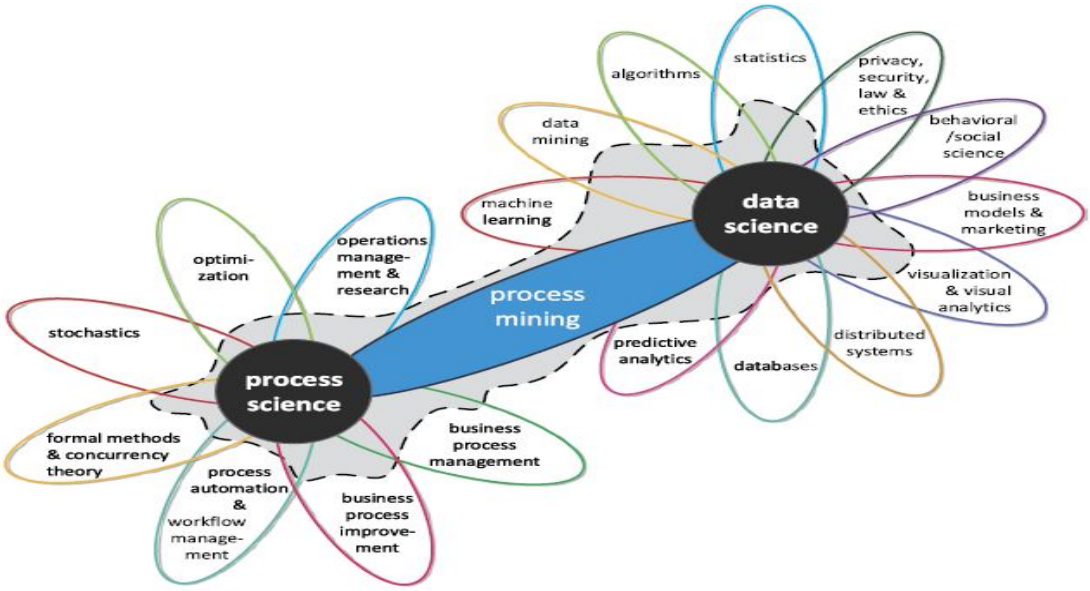

Abbildung 1.4: Verbindung zwischen Data-Science und Process-Science (Aalst 2016, S.16)

Process-Mining kann folglich als die fehlende Verbindung zwischen Data-Science und Process-Science gesehen werden (Abbildung 1.4). Process-Mining versucht eine Verbindung zwischen Event-Daten, d.h. das beobachtete Verhalten, und den Prozessmodellen, welche per Hand erstellt oder durch Algorithmen automatisch entdeckt worden sind, herzustellen (vgl. Aalst 2016, S.17).

Data-Science fokussiert sich sehr auf die Daten. Es werden viele Kennzahlen und Erkenntnisse gewonnen, aber diese werden nicht in Verbindung mit Problemen/Prozessen in der realen Welt gebracht. Process-Science hingegen vernachlässigt die echten Daten und erschafft eine idealisierte Welt, die in den meisten Fällen in der Art nicht existiert.

Process-Mining arbeitet hingegen auf realen Daten und kann, wie in Abbildung 1.5 zu erkennen, in drei Teilbereiche untergliedert werden:

- Process-Discovery beschäftigt sich mit der Erstellung eines Prozess-Modells auf Basis eines Eventlogs.
- Process-Conformance überprüft die Übereinstimmung zwischen einem Eventlog und einem Prozess-Modell, um beispielsweise Engpässe zu identifizieren.
- Process-Enhancement erweitert ein bestehendes Prozess-Modell um zusätzliche Informationen.

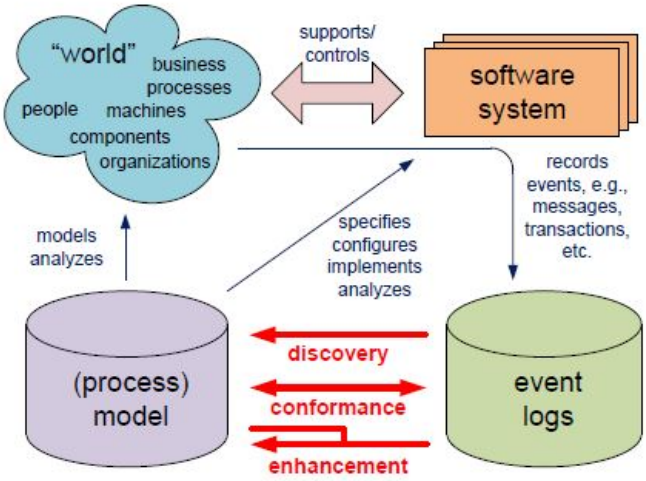

Abbildung 1.5: Techniken des Process-Minings

Die Fragen, die mit Hilfe von Process-Mining beantwortet werden sollen, können in vier verschiedene Perspektiven unterteilt werden (vgl. Aalst 2016, S. 34):

- Die Kontrollflussperspektive fokussiert sich auf die Anordnung der Aktivitäten. Ziel dieser Perspektive ist es, eine gute Beschreibung aller möglichen Pfade eines Prozesses in einer visualisierten Form, wie z.B. Petri-Netze oder BPNM-Modelle, zu erstellen.
- Die Organisationsperspektive beschäftigt sich mit den am Prozess beteiligten Ressourcen und untersucht, wie diese miteinander verbunden ist. Ziel hierbei ist es,

den Mitarbeitern Rollen oder organisatorische Einheiten zuzuordnen oder die Abhängigkeit zwischen den Ressourcen in Form eines Social-Networks darzustellen.

- Die Zeitperspektive untersucht den Zeitpunkt und die Häufigkeit von Events. Wenn ein Event einen Zeitstempel trägt, ist es möglich Engpässe zu entdecken oder Service-Level zu messen.
- Die Caseperspektive untersucht Attribute, die nicht mit einem einzelnen Event, sondern mit einem Case verbunden sind. Ein Case kann als konkrete Instanz eines Prozesses verstanden werden. Handelt es sich beispielsweise um einen Bestellprozess, wären Informationen wie der Name des Lieferanten, Preis der Bestellung oder die bestellten Teile von Interesse.

Process-Mining kann auch mit dem sogenannten BPM-Life-Cycle, zu sehen in Abbildung 1.6, in Verbindung gebracht werden. In der Design-Phase wird ein Prozess-Modell erstellt, welches in der Konfigurations-/Implementationsphase in das laufende System eingebettet wird.

Sobald das System den Prozess unterstützt, startet die Monitoring-Phase, in welcher die Prozesse überwacht werden, um eventuelle Änderungen zu erkennen. Manche dieser Veränderungen können direkt in der Adjustment-Phase behoben werden. Sollten diese Veränderungen aber gravierender sein, was in der Diagnose-Phase festgestellt wird, muss der Prozess neugestaltet werden; man startet folglich in eine neue Iteration mit der Design-Phase (vgl. Aalst 2016, S. 31).

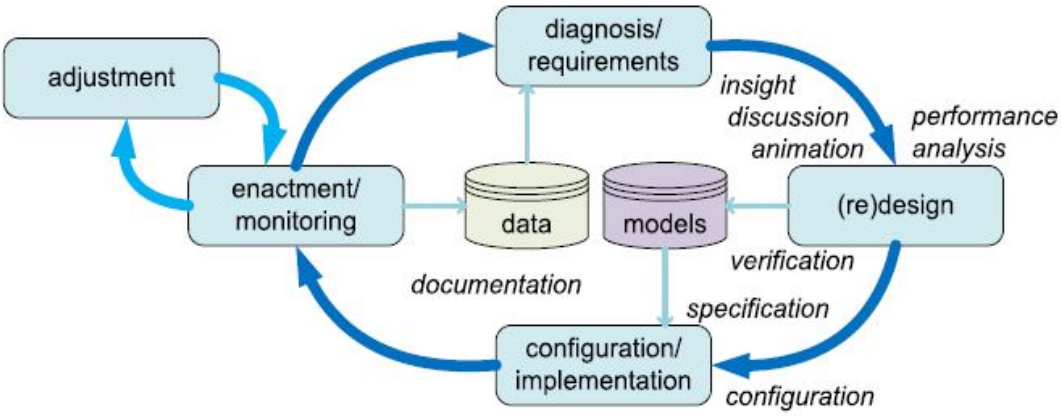

Abbildung 1.6: BPM-Life-Cycle (Aalst 2016, S.31)

Process-Mining wird in diesem Kontext genutzt, um die Prozesse in der Designphase zu entdecken. Die in der Monitoring-Phase gesammelten Event-Daten können genutzt werden, um ein besseres Verständnis des wirklichen Prozesses zu erhalten. Anschließend könnten die Abweichungen analysiert (Process-Conformance) oder das Modell (Process-Enhancement) verbessert werden.

Aufbau der Bachelorarbeit

Die Arbeit gliedert sich in einen theoretischen Teil, welcher den bisherigen Stand der Forschung im Gebiet Process-Mining untersucht, und einen praktischen Teil, in dem der Entwicklungsprozess der Process-Mining-Bibliothek beschrieben wird.

In Abschnitt 2.1 wird das Thema der Datenbeschaffung beschrieben. Die Datenbeschaffung nimmt eine sehr wichtige Rolle im Process-Mining ein, da die Daten der Ausgangspunkt für jede Anwendung von Process-Mining sind. Weisen die Daten keine gute Qualität auf, können die Ergebnisse von Process-Mining ebenfalls keine gute Qualität erreichen. Im Kontext der Datenbeschaffung wird erläutert, aus welchen Quellen man Daten beziehen kann, wie sie anschließend zu einem Eventlog weiterverarbeitet werden können und anhand von welchen Merkmalen die Qualität der Daten gemessen werden kann.

Als nächstes wird in 2.2 der Teilbereich Process-Discovery anhand des α-Algorithmus eingeführt. Dabei werden zunächst Petri-Netze als eine Form der Visualisierung von Prozess-Modellen vorgestellt. Anschließend wird der α-Algorithmus anhand eines Beispiels Schritt für Schritt durchgeführt. Außerdem werden die Einschränkungen des α-Algorithmus besprochen. Zuletzt werden in diesem Abschnitt Kriterien erläutert, mit denen man die Qualität eines Prozess-Modells beschreiben kann.

Anschließend werden in Abschnitt 2.3 die Eigenschaften von Process-Discovery-Algorithmen vorgestellt und fortgeschrittene Algorithmen, wie z.B. der $\alpha+$-Algorithmus, Heuristic Mining oder Genetic Mining sowie deren Einsatzgebiet besprochen.

Abschnitt 2.4 beschäftigt sich mit einem weiteren Teilbereich von Process-Mining, der Process-Conformance. Es werden drei verschiedene Techniken erläutert, mit denen die Übereinstimmung von Eventlog und Prozess-Modell geprüft werden kann.

Process-Enhancement, als letzter Teilbereich von Process-Mining, wird in Abschnitt 2.5 besprochen. Dabei werden die Möglichkeiten der Organisations-, Zeit- und Caseperspektive vorgestellt.

Der letzte Abschnitt des theoretischen Teils gibt eine kurze Einführung in den Umgang mit dem Process-Mining-Tool ProM.

Der Aufbau des praktischen Teils orientiert sich an den Phasen eines Softwareentwicklungs-Prozesses.

Abschnitt 3.1. beschreibt die Anforderungen an die Bibliothek. Anschließend wird in Abschnitt 3.2 der Entwurf der Bibliothek anhand eines Komponenten- und Sequenzdiagramms erläutert. Als Nächstes folgt in Abschnitt 3.3 die Implementation der vier Komponenten der Bibliothek. Dabei wird unter anderem XES als Format für Eventlogs und PNML als Format für Petri-Netze eingeführt. In Abschnitt 3.4 wird beschrieben, wie die Bibliothek getestet werden konnte und welche Resultate dabei erreicht wurden. Danach wird in Abschnitt 3.5 auf das Deployment der Bibliothek eingegangen. Zum Schluss wird in Abschnitt 3.6 ein Ausblick gegeben, inwiefern die Bibliothek erweiterbar ist und welche Erweiterungen in Zukunft vorgenommen werden könnten.

Am Ende dieser Arbeit werden die wichtigsten Punkte von Process-Mining zusammengefasst und ein Ausblick auf nachfolgende Arbeiten gegeben.

Ziel und Motivation der Bachelorarbeit

Die Motivation dieser Bachelorarbeitet besteht in erster Linie darin, Process-Mining in beliebigen Informationssystemen zu etablieren, um Erkenntnisse über komplexe oder sogar noch unerforschte Prozesse zu erlangen, welche sich sogar über System- und/oder Unternehmensgrenzen erstrecken können. Diese Erkenntnisse können anschließend verwendet werden, um die zugrundeliegenden Prozesse zu verbessern.

Ziel dieser Bachelorarbeit ist es, einen verständlichen Überblick über das komplexe Thema Process-Mining zu gewähren und zusätzlich eine prototypische Bibliothek zu entwickeln, mit der die verschiedensten Techniken des Process-Minings angewendet werden können. Der Fokus liegt dabei auf der Erstellung von standardisierten Eventlogs, welche die Basis für die Anwendung von Process-Mining darstellen, und auf den Teilbereich Process-Discovery, da die anderen beiden Teilbereiche auf diesen aufbauen. Nichtsdestotrotz werden Process-Conformance und Process-Enhancement erläutert, um die zukünftigen Möglichkeiten dieser Bibliothek aufzuzeigen.

2 State of the Art

Dieses Kapitel stellt die Grundlagen des Process-Minings vor, um eine Basis für die Entwicklung der Bibliothek zu schaffen. Zuerst wird die Prägnanz der Datenbeschaffung erörtert. Anschließend werden die drei Teilbereiche Process-Discovery, Process-Conformance und Process-Enhancement vorgestellt. Zuletzt wird ein Einblick in das bestehende Process-Mining-Tool Prom gegeben.

2.1 Datenbeschaffung

Process-Mining ist ohne korrekte Eventlogs nicht durchführbar. In diesem Abschnitt werden die Informationen beschrieben, die in jedem Eventlog vorhanden sein sollten. Dabei sollte man beachten, dass die Informationen abhängig von der verwendeten Process-Mining-Technik und dem gewünschten Ergebnis variieren können. Die Herausforderung bei der Datenbeschaffung besteht darin, dass die Daten aus verschiedenen Datenquellen extrahiert werden müssen. Bei der Zusammenführung dieser Daten nehmen sowie die Syntax als auch die Semantik eine wichtige Rolle ein (vgl. Aalst 2016, S.125).

2.1.1 Datenquellen

Eine Datenquelle kann eine einfache Excel-Datei, eine Datenbank oder ein ERP-System sein. In der Realität sind die Daten aber nicht in einer strukturierten Datenquelle vorhanden, sondern über mehrere Datenquellen verteilt und die Zusammenführung dieser Daten verursacht hohen Aufwand.

Eine vollständige SAP-Implementation hat typischerweise über 10.000 Tabellen, welche aus technischen oder organisatorischen Gründen aufgeteilt werden könnten. Darüber hinaus könnten Legacy-Systeme existieren, die noch wichtige Daten beinhalten, oder Systeme, die nur in bestimmten Abteilungen ihren Einsatz finden und nicht in die

Systemlandschaft des gesamten Unternehmens integriert wurden sind. Untersucht man außerdem Supply-Chain-Prozesse, so sind die Daten sogar über mehrere Unternehmen verteilt (vgl. Aalst 2016, S.126).

Neben der Verteilung der Daten, ist die Unstrukturiertheit von Daten ein weiteres Problem. Daten könnten z.B. in Form von Emails, PDF-Dokumenten, Bildern oder eingescannten Texten vorliegen. Die Komplexität des Themas Datenbeschaffung ist hoch, sodass diese nur gezielt anhand bestimmter vorher definierter Fragestellungen durchgeführt werden sollte.

Um diesen Problemen entgegenzuwirken, sollten die Daten per ETL-Prozess in ein Data-Warehouse geladen werden. Der Begriff ETL bedeutet:

- Extract: Die Daten müssen aus verschiedenen Datenquellen extrahiert werden.
- Transform: Die Daten müssen transformiert werden, sodass sie für die weitere Verarbeitung verwendbar sind, indem sie z.B. vereinheitlicht werden. Dies beinhaltet den Umgang mit semantischen und syntaktischen Problemen. Unterschiedliche Datenquellen können unterschiedliche Schlüssel verwenden. Beispielsweise kann in der einen Datenquelle ein Patient durch seinen Nachnamen und Geburtstag identifiziert werden, während in der anderen Datenquelle die Sozialversicherungsnummer verwendet wird. Eine Datenquelle könnte das Datumsformat 31-12-2019 verwenden, wohingegen in der anderen Datenquelle dieses Datum als 2019/12/31 abgespeichert wird.
- Load: Die Daten müssen in separates System, z.B. ein Data-Warehouse, geladen werden.

Ein Data-Warehouse ist eine logische Ansammlung der operationalen Daten eines Unternehmens. Das Erstellen eines Data-Warehouse erzeugt dabei keine neuen Daten, sondern es hat als Ziel Daten aus verschiedenen Quellen zu vereinheitlichen.

Die Existenz eines Data-Warehouse vereinfacht den Vorgang des Process-Minings zwar, ist aber nicht zwingend notwendig. Es ist wichtig, dass die Daten extrahiert und in ein Eventlog umgewandelt werden können. Ein typisches Format um Eventlogs zu speichern ist XES (eXtensible Event Stream), welches im Abschnitt 3.3.2 genauer erläutert wird.

Bei der Extraktion der Daten ist die Auswahl der relevanten Daten von großer Wichtigkeit. Dabei wird zwischen zwei Filterungsstufen unterschieden. Bei der ersten, gröberen

Stufe werden nur die Events herausgefiltert, die zu den ausgewählten Prozessen gehören.

Die zweite, feinere Stufe filtert anschließend nur bestimmte Events anhand ihrer Attribute oder ihrer Häufigkeit heraus. Basierend auf dem gefilterten Log können die verschiedenen Teilgebiete des Process-Minings angewendet werden: Process-Discovery, Process-Conformance und Process-Enhancement (vgl. Aalst 2016, S. 127).

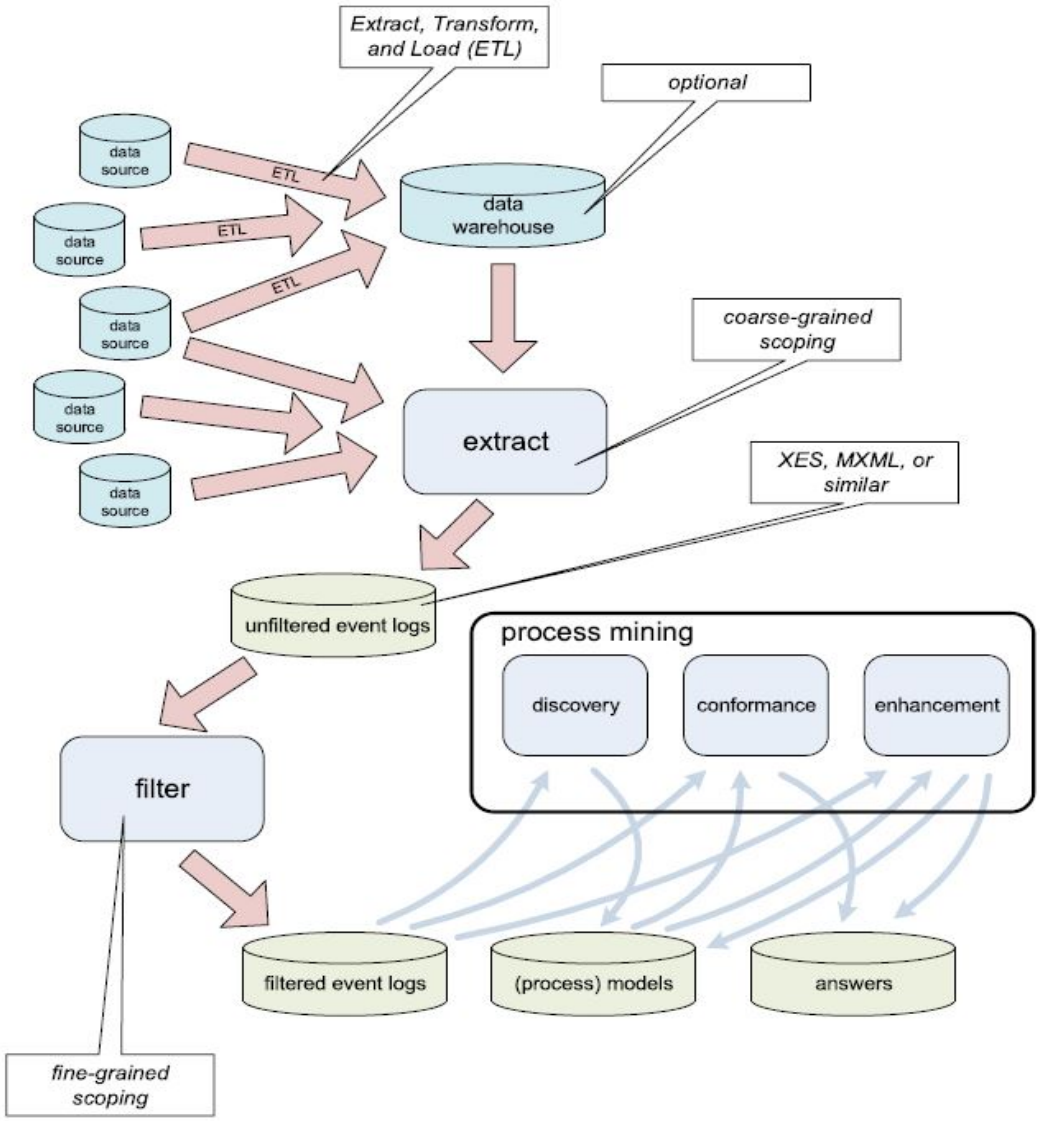

Abbildung 2.1: Workflow-Übersicht, um von Daten aus verschiedenen Quellen zu Process-Mining-Ergebnissen zu gelangen (Aalst 2016, S.126)

2.1.2 Eventlogs

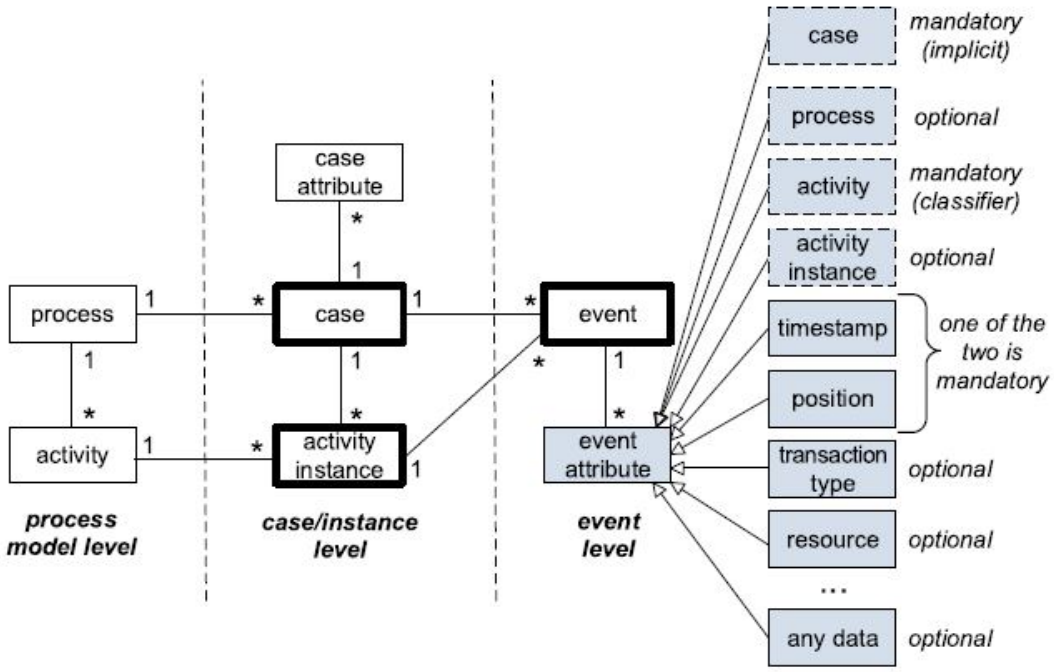

Abbildung 2.2: UML-Klassendiagramm Eventlog (Aalst 2016, S.147)

Um den Aufbau eines Eventlogs zu verstehen, kann das Klassendiagramm in Abbildung 2.2 betrachtet werden:

1. Jeder Prozess besteht aus einer beliebigen Anzahl an Aktivitäten, aber jede Aktivität ist genau einem Prozess zugeordnet. Ein Prozess wäre beispielsweise die Bearbeitung eines Schadensersatzantrages, eine Aktivität analog dazu *Registriere Antrag* oder *Antrag ablehnen.*
2. Jeder Case gehört zu einem Prozess. Ein Case ist eine Prozessinstanz.
3. Jede Aktivitätsinstanz bezieht sich auf genau eine Aktivität.
4. Jede Aktivitätsinstanz gehört genau zu einem Case.
5. Jedes Event bezieht sich auf genau einen Case.
6. Jedes Event gehört genau zu einer Aktivitätsinstanz. Für die gleiche Aktivitätsinstanz können mehrere Events existieren.

7. Jedes Case-Attribut bezieht sich auf genau einen Case; jedes Attribut hat einen Namen und einen Wert, wie z.B. *(Name, Peter Müller)*.

8. Jedes Event-Attribut gehört genau zu einem Event und setzt sich aus Name und Wert zusammen, wie z.B. *(Kosten, 200€)*.

9. Es existieren verschiedene Unterkategorien von Event-Attributen, wie z.B. der Zeitstempel, die ausführende Ressource oder die Kosten eines Events.

10. Ein Event benötigt mindestens drei Attribute: Den Case, die Aktivität und die Position innerhalb eines Case oder den Zeitstempel.

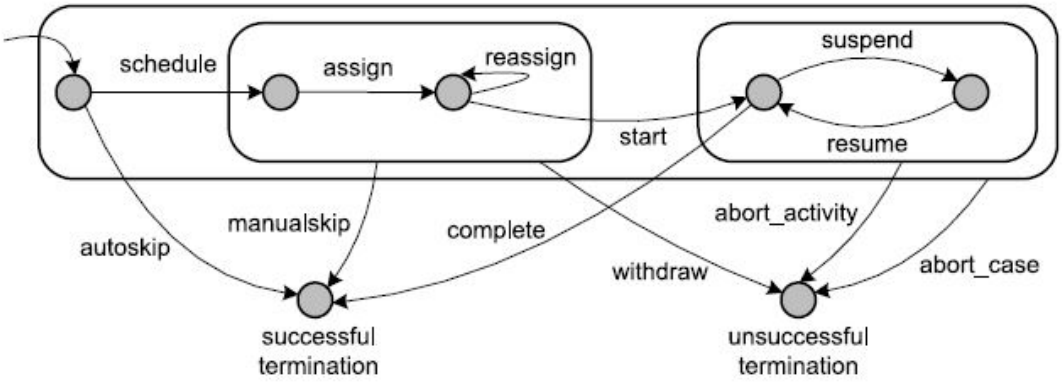

Abbildung 2.3: Standard Transactional-Life-Cycle-Model (Aalst 2016, S.131)

Ein weiteres wichtiges Attribut ist der Transaktionstyp. Dieses Attribut eines Events teilt die dazugehörige Aktivitätsinstanz in mehrere kleine Abschnitte ein und ermöglicht dadurch, dass mehrere Events einer Aktivitätsinstanz zugeordnet werden können.

Oft werden einer Aktivitätsinstanz nur vollständig abgeschlossene Events zugewiesen. Dadurch stehen Aktivitätsinstanz und Event in einer 1-zu-1-Beziehung und es würde keine Unterscheidung mehr zwischen Aktivitätsinstanz und Event nötig sein. Sobald Events in Start- und End-Events unterteilt werden, wird der Unterschied zwischen Aktivitätsinstanz und Event deutlich. Zum Beispiel könnte einer Aktivitätsinstanz *Überprüfe Schadensersatzantrag* die beiden Events *Überprüfe Schadensersatzantrag-Start* und *Überprüfe Schadensersatzantrag-Ende* zugewiesen werden. Die Verwendung des Transaktion-Typ-Attributes ermöglicht das Messen von Durchlaufzeiten und die Identifizierung von Eng-

pässen. In Abbildung 2.4 sind Beispiele zu sehen, wie einer Aktivitätsinstanz mehrere Events mit Transaktionstyp-Attribut zugewiesen werden können.

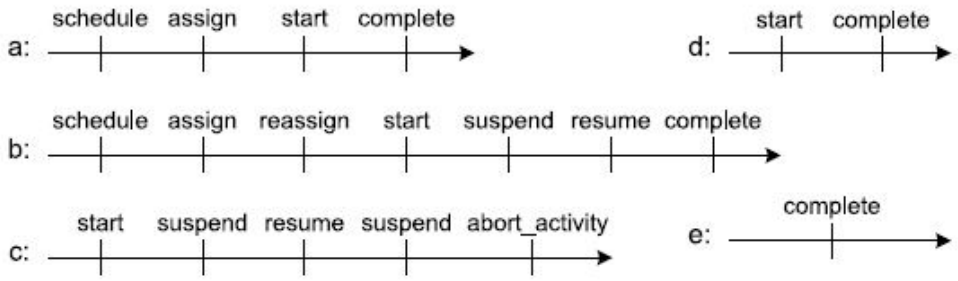

Abbildung 2.4: Beispiel Events mit Transaktionstyp-Attribut (Aalst 2016, S.131)

Abbildung 2.5 zeigt einen Ausschnitt eines konkreten Eventlogs, welches zu dem Prozess *Schadensersatzantrag* gehört. Zu sehen sind insgesamt vier Cases, die unterschiedlich viele Events beinhalten. Die Events besitzen eine ID, sind geordnet und setzen sich aus den Attributen Timestamp, Activity, Resource und Cost zusammen.

Dieses Eventlog enthält alle relevanten Informationen, aber kann bei zu vielen Cases schnell unübersichtlich werden. Aus diesem Grund wurden die sogenannten Simple-Eventlogs entwickelt. Ein Simple-Eventlog besteht aus einem Multi-Set von Traces, wobei ein Trace eine Sequenz von Aktivitäten ist.

Ein Beispiel für ein Simple-Eventlog wäre [(a, b, c, d)^3, (a, c, b, d)^2, (a, e, d)], welcher aus 6 Cases und 3x4 + 2x4 + 1x3 = 23 Events besteht. In einem Simple-Eventlog sind keine Attribute, wie z.B. Zeitstempel oder Ressource, zu erkennen (vgl. Aalst 2016, S. 136).

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	...
...	...	...	...	...	...	...

Abbildung 2.5: Ausschnitt eines Eventlogs (Aalst 2016, S.129)

Die Erstellung von Eventlogs kann sehr anspruchsvoll sein. Im Folgenden werden die größten Herausforderungen vorgestellt (vgl. Aalst 2016, S. 142-144 und Bose u. a. 2013, S. 3-13):

- Zusammenhang: Events im Eventlog müssen einem Case zugeordnet werden. Diese simple Anforderung kann schnell zu Problemen führen, insbesondere dann, wenn

die Daten aus verschiedenen Systemen oder sogar aus verschiedenen Unternehmen stammen.

- Zeitstempel: Die Events innerhalb eines Eventlogs müssen geordnet sein, wofür das Attribut Zeitstempel verwendet werden kann. Bei Zeitstempeln existieren mehrere Probleme, wie beispielsweise unterschiedliche Zeitzonen, lokale Uhren oder ein unterschiedlicher Detaillierungsgrad.

- Snapshots: Eventlogs sind nur eine Erfassung eines Zustandes zu einem bestimmten Zeitpunkt. In einem Eventlog könnten Cases vorhanden sein, die vor Erfassung der Daten gestartet oder noch nicht abgeschlossen sind. Unvollständige Cases können daran erkannt werden, dass ihre durchschnittliche Laufzeit im Vergleich zu anderen Cases gering ist oder dass ihnen Start- oder End-Aktivitäten fehlen. Diese unvollständigen Cases sollten aus dem Eventlog ausgeschlossen werden.

- Auswahl der relevanten Daten: Die Daten könnten beispielweise über mehrere tausend Tabellen verteilt liegen und es bedarf an Expertenwissen, um die richtigen Daten für gegebene Fragestellungen auszuwählen.

- Granularität: Die Granularität der Events muss den Stakeholdern angepasst sein, die an dem Prozess interessiert sind. Sehr detaillierte Low-Level-Events könnten die Stakeholder überfordern und sollten daher zu grobgranulareren Events zusammengefasst werden.

- Case-Heterogenität: Viele der heutigen Prozesse sind auf Flexibilität ausgelegt. Zwar ist es wünschenswert, alle möglichen Szenarien eines Prozesses in einem Eventlog vertreten zu haben, aber dies bringt ein großes Problem mit sich. Bei einer großen Anzahl von möglichen Szenarien könnten die erzeugten Prozess-Modelle unübersichtlich werden.

- Datenvolumen: Process-Mining hat in der Ära von *Big Data* immer mehr an Bedeutung gewonnen, da aus der großen Anzahl von vorhandenen Daten wertvolle Informationen extrahiert werden sollen. Gleichzeitig sind die Process-Mining-Techniken nicht in der Lage, mit solchen Datenmengen umzugehen. Daher sollte die Entwicklung effizienter, skalierbarer und verteilter Algorithmen vorangetrieben werden.

2.1.3 Datenqualität

Die Datenqualität ist von äußerster Wichtigkeit für ein erfolgreiches Process-Mining. Sollten Eventdaten fehlen oder ihre Richtigkeit in Frage gestellt sein, sind die Ergebnisse von Process-Mining nur von geringem Wert.

Ein Eventlog besteht grundsätzlich aus drei Hauptentitäten: Case, Aktivitätsinstanz und Event. Auf Entitätslevel existieren drei Probleme:

1. Die Entität fehlt im Log, aber sie ist in der Realität aufgetreten.
2. Die Entität fehlt in der Realität, wurde aber fälschlicherweise in das Eventlog mit aufgenommen.
3. Die Entität ist im Log versteckt, d.h., sie verbirgt sich hinter einer größeren Ansammlung von Daten.

Auf Attributsebene hingegen existieren folgende Probleme:

1. Ein Attribut fehlt, z.B. wurde der Zeitstempel nicht aufgenommen.
2. Ein Attribut ist fehlerhaft, z.B. wurde ein Event dem falschen Case zugeordnet.
3. Ein Attribut ist zu unpräzise, z.B. enthält der Zeitstempel nur das Datum, aber nicht die genaue Uhrzeit.

Um all diesen Datenqualitätsproblemen entgegenzuwirken, wurden 12 Guidelines für Logging von Daten entwickelt (vgl. Aalst 2015, S.7-8), welche beispielsweise folgende Punkte beinhalten:

- Die Attribute sollten eine klar definierte Semantik besitzen, die von allen Stakeholdern verstanden wird.
- Die Events sollten, wenn möglich, Transaktionstyp-Attribute (Start, Ende, pausiert etc.) speichern.
- Regelmäßig automatisierte Konsistenz- und Korrektheitsprüfungen sollten durchgeführt werden, um die Qualität des Eventlogs zu gewährleisten.
- Private Daten sollten entfernt werden, ohne dabei wichtige Zusammenhänge zu verlieren. Die Daten könnten beispielweise mit Hilfe von Hashing anonymisiert werden.

Die zu entwickelnde Bibliothek zur generischen Anwendung von Process-Mining, welche in Kapitel 3 vorgestellt wird, soll zur Datenqualität beitragen. Beispielsweise könnte die Bibliothek wichtige Transform-Schritte bezüglich des Attributes Zeitstempel umsetzen, um die Homogenität der Daten sicherzustellen. Darüber hinaus könnte es die Eventlogs auf fehlende Attribute, die zwingend notwendig sind, überprüfen und dafür sorgen, dass die Eventlogs vollständig sind.

2.2 Process-Discovery: α-Algorithmus

Process-Discovery ist eine der herausforderndsten Aufgaben von Process-Mining. Basierend auf einem Eventlog wird ein Prozess-Modell konstruiert, welches das im Eventlog beobachtete Verhalten abbilden soll.

Process-Discovery beschäftigt sich mit der Kontrollflussperspektive, d.h. die Anordnung der Aktivitäten steht im Vordergrund. Ein Process-Discovery-Algorithmus ist per Definition eine Funktion, die ein Eventlog auf ein Prozess-Modell abbildet, sodass das Modell repräsentativ für das im Eventlog beobachtete Verhalten steht. Die Definition lässt offen, welche Notation für die Repräsentation (z.B. BPMN, EPK, YAWL oder Petri-Netze) gewählt werden soll (vgl. Aalst 2016, S. 163). Das entstandene Prozess-Modell kann bei Bedarf auch im Nachhinein in andere Notationen transformiert werden (vgl. Aalst 2016, S. 165).

Dieser Abschnitt führt Petri-Netze als eine Darstellungsmöglichkeit von Prozess-Modellen ein und beschreibt den α-Algorithmus, welcher die Grundgedanken hinter vielen Process-Mining-Algorithmen verdeutlicht.

2.2.1 Petri-Netze

Im weiteren Verlauf dieser Arbeit werden Petri-Netze als Darstellungsmöglichkeit gewählt. Der Grund dafür ist, dass Petri-Netze simpel und grafisch sind, trotzdem erlauben sie die Modellierung von Parallelität, Auswahlmöglichkeiten und Iterationen (vgl. Aalst 2016, S. 165).

Ein Petri-Netz ist per Definition ein Triplett N = (P, T, F):
Das P steht für eine endliche Menge an Places, das T für eine endliche Menge an Transitions und das F für eine Flow-Relation, welche wiederum eine Menge aus gerichteten Arcs ist. Ein Arc verbindet einen Place mit einer Transition oder eine Transition mit einem Place.

Transitions werden als Quadrate dargestellt und stehen für die Aktivitäten, welche in dem zu modellierenden Prozess stattfinden. Transition bedeutet wörtlich Wandel oder Übergang, bezogen auf Petri-Netze sorgen die Transitions für den Übergang von einem Zustand in den Nächsten (vgl. Reisig 2010, S. 22-24).

Diese Zustände werden auch als Places bezeichnet und als Kreise dargestellt. Die Verbindungen zwischen Transitions und Places, die sogenannten Arcs, werden als Pfeile dargestellt. In Abbildung 2.6 ist ein Beispiel eines Petri-Netzes zu sehen.

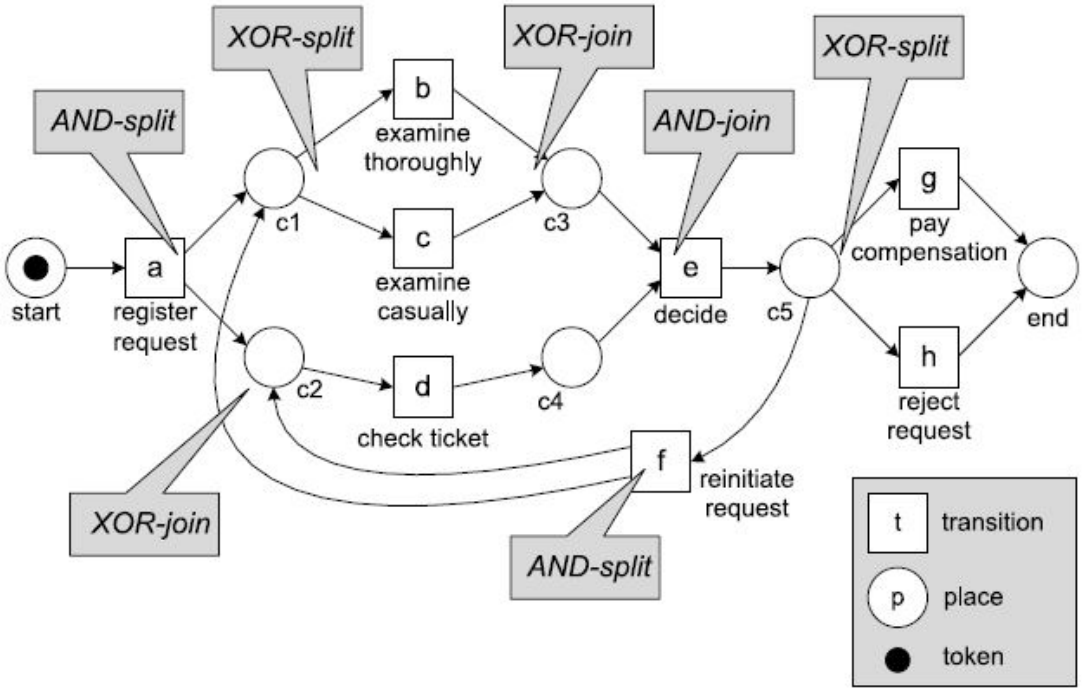

Abbildung 2.6: Beispiel Petri-Netz (Aalst 2016 S. 60)

Bei diesem Beispiel handelt es sich sogar um ein markiertes Petri-Netz. Ein solches Petri-Netz ist per Definiton ein Triplett N, M = (P, T, F). M steht für Marking und ist ein Multi-Set über die Places. Im Falle des Beispiels wäre das Marking [start] und wird durch einen schwarzen Punkt, ein sogenanntes Token, dargestellt. Eine Transition wird nur dann aktiviert, wenn jeder seiner Input-Places ein Token beinhaltet. Eine aktivierte Transition kann auslösen, wodurch je ein Token von seinen Input-Places konsumiert und je ein Token für seine Output-Places produziert wird. Die Transition a wäre mit dem Marking [start] aktiviert. Löst diese Transition aus, würde ein Token konsumiert und zwei Token produziert, was in dem Marking [c1, c2] resultiert (vgl. Aalst 2016, S.61).

Die formalisierte Beschreibung des gezeigten Petri-Netzes würde wie folgt aussehen:

- P = {start, c1, c2, c3, c4, c5, end}
- T = {a, b, c, d, e, f, g, h}

- 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)}

2.2.2 α-Algorithmus

Grundidee

Der α-Algorithmus ist ein Beispiel für einen Process-Discovery-Algorithmus. Dieser Algorithmus war einer der ersten Process-Discovery-Algorithmen, der mit Parallelität umgehen konnte. Nichtsdestotrotz hat der α-Algorithmus einige Einschränkungen, welche in Abschnitt 2.3.2 erläutert werden.

Die Grundidee des α-Algorithmus besteht darin, bestimmte Muster in dem Eventlog zu erkennen. Wenn eine Aktivität a gefolgt wird von einer Aktivität b, aber b nicht gefolgt wird von a, so wird angenommen, dass eine kausale Abhängigkeit zwischen a und b besteht. Um diese Abhängigkeit widerzuspiegeln, sollte das dazugehörige Petri-Netz ein Place beinhalten, welcher a mit b verbindet.

Es wird zwischen vier Ordnungsrelationen unterschieden, um die relevanten Muster in einem Eventlog zu erfassen:

Definition 6.3 (Log-based ordering relations) Let L be an event log over $\mathscr{A}$, i.e., $L \in \mathbb{B}(\mathscr{A}^*)$. Let $a, b \in \mathscr{A}$.

- $a >_L b$ if and only if there is a trace $\sigma = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ and $i \in \{1, \ldots, n-1\}$ such that $\sigma \in L$ and $t_i = a$ and $t_{i+1} = b$;
- $a \rightarrow_L b$ if and only if $a >_L b$ and $b \not>_L a$;
- $a \#_L b$ if and only if $a \not>_L b$ and $b \not>_L a$; and
- $a \|_L b$ if and only if $a >_L b$ and $b >_L a$.

Abbildung 2.7: Eventlog-basierte Ordnungsrelationen (Aalst 2016, S.168)

Als Input für den α-Algorithmus wird ein Simple-Eventlog verwendet. Ziel ist es, ein Petri-Netz zu erzeugen, welches alle im Eventlog enthaltenen Traces nachspielen kann. Als Basis für die Erläuterungen dieses Kapitels wird folgendes Simple-Eventlog angenommen:

L_1 = [(a, b, c, d)^3, (a, c, b, d)^2, (a, e, d)]

Die erste Ordnungsrelation beinhaltet alle Paare von Aktivitäten, die direkt aufeinander folgen:

$>L_1$ = {(a, b), (a, c), (a, e), (b, c), (c, b), (b, d), (c, d), (e, d) }

Die zweite Ordnungsrelation beinhaltet alle Paare in einer Kausalität-Relation, d.h c -> d, da d direkt c folgt, aber nicht andersrum:

$->L_1$ = {(a, b), (a, c), (a, e), (b, d), (c, d), (e, d) }

Die dritte Ordnungsrelation beinhaltet alle Paare, die keine direkte Verbindung zueinander haben:

$\#L_1$ = {(a, a), (a, d), (b, b), (b, e), (c, c), (c, e), (d, a), (d, d), (e, b), (e, c), (e, e) }

Die vierte Ordnungsrelation beinhaltet alle Paare, die in Parallelitäts-Relation zueinanderstehen:

$||L_1$ = { (b, c), (c, b) }

Eine kompakte Darstellungsmöglichkeit dieser Ordnungsrelationen ist die sogenannte *Footprint-Matrix*, welche in Abbildung 2.8 zu sehen ist.

	a	*b*	*c*	*d*	*e*
a	$\#_{L_1}$	$\rightarrow_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$	$\rightarrow_{L_1}$
b	$\leftarrow_{L_1}$	$\#_{L_1}$	$\parallel_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$
c	$\leftarrow_{L_1}$	$\parallel_{L_1}$	$\#_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$
d	$\#_{L_1}$	$\leftarrow_{L_1}$	$\leftarrow_{L_1}$	$\#_{L_1}$	$\leftarrow_{L_1}$
e	$\leftarrow_{L_1}$	$\#_{L_1}$	$\#_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$

Abbildung 2.8: Footprint-Matrix (Aalst 2016, S.168)

Anhand der Footprint-Matrix können bestimmte Muster abgeleitet werden:

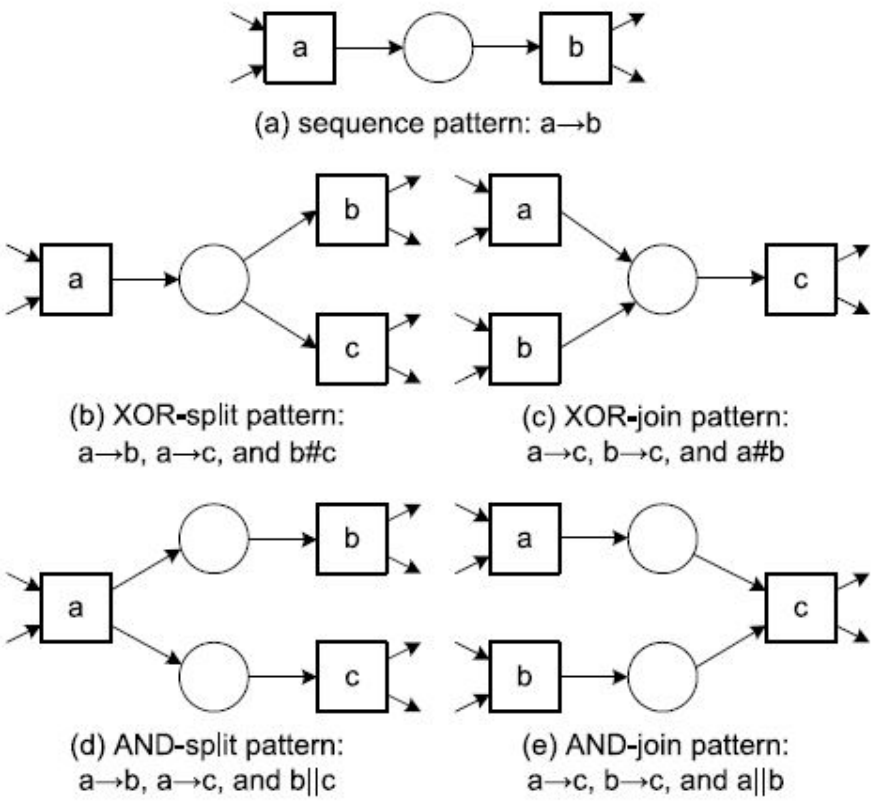

Abbildung 2.9: Patterns und ihre Footprints in einem Eventlog (Aalst 2016, S.169)

Hat man beispielsweise nach a die Wahl, ob b oder c folgen soll, so würde das Eventlog die Footprints a -> b, a -> c und b#c hinterlassen.

Algorithmus

Nachdem die Grundidee und weitere Beispiele vorgestellt wurden sind, folgt die Beschreibung des α-Algorithmus (vgl. Aalst u. a. 2004, S. 8-10). Die Definition des α-Algorithmus lautet:

Definition 6.4 (α-algorithm) Let L be an event log over $T \subseteq \mathscr{A}$. $\alpha(L)$ is defined as follows:

1. $T_L = \{t \in T \mid \exists_{\sigma \in L}\ t \in \sigma\}$,
2. $T_I = \{t \in T \mid \exists_{\sigma \in L}\ t = first(\sigma)\}$,
3. $T_O = \{t \in T \mid \exists_{\sigma \in L}\ t = last(\sigma)\}$,
4. $X_L = \{(A, B) \mid A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \wedge \forall_{a \in A} \forall_{b \in B}\ a \rightarrow_L b \wedge \forall_{a_1, a_2 \in A}\ a_1 \#_L a_2 \wedge \forall_{b_1, b_2 \in B}\ b_1 \#_L b_2\}$,
5. $Y_L = \{(A, B) \in X_L \mid \forall_{(A', B') \in X_L} A \subseteq A' \wedge B \subseteq B' \Longrightarrow (A, B) = (A', B')\}$,
6. $P_L = \{p_{(A,B)} \mid (A, B) \in Y_L\} \cup \{i_L, o_L\}$,
7. $F_L = \{(a, p_{(A,B)}) \mid (A, B) \in Y_L \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A, B) \in Y_L \wedge b \in B\} \cup \{(i_L, t) \mid t \in T_I\} \cup \{(t, o_L) \mid t \in T_O\}$, and
8. $\alpha(L) = (P_L, T_L, F_L)$.

Abbildung 2.10: Formale Beschreibung des α-Algorithmus (Aalst 2016, S.171)

Die Schritte 4 und 5 stellen dabei die größten Herausforderungen dar. Die einzelnen Schritte werden nun anhand des Simple-Eventlogs L_1 = [(a, b, c, d)^3, (a, c, b, d)^2, (a, e, d)] durchgespielt:

1. Überprüfe, welche Events im Eventlog vorhanden sind. Diese Events werden die Transitions des Petri-Netzes.

T_L = {a, b, c, d, e}

2. Ermittle alle Events, die am Anfang eines Traces stattgefunden haben.

T_I = {a}

3. Ermittle alle Events, die am Ende eines Traces stattgefunden haben.

T_O = {e}

4. Erzeuge alle non-maximal Places. Ein non-maximal Place ist ein Tupel (A, B), bestehend aus zwei nicht-leeren Teilmengen von T_L. Diese Teilmengen müssen zwei Bedingungen erfüllen:

1. Jedes Element aus A muss mit jedem Element aus B in einer Kausalitäts-Relation (->) stehen.
2. Die Elemente aus A dürfen keine Verbindung (#) zueinander haben. Die Elemente aus B dürfen keine Verbindung (#) zueinander haben.

X_L = { {(a), (b)}, {(a), (c)}, {(a), (e)}, {(a), (b, e)}, {(a), (c, e)}, {(b), (d)}, {(c), (d)}, {(e), (d)}, {(b, e), (d)}, {(c, e), (d)} }

5. Würde man für jedes Element aus X_L ein Place erzeugen, würden zu viele Places existieren und der Übersichtlichkeit des Prozess-Modells schaden. Daher werden nur alle maximal Places verwendet. Um diese zu erhalten, wird überprüft, ob die Elemente A und B eines non-maximal Places (A, B) Teilmengen von einem anderen non-maximal-Places (Á,Ḃ) sind. Falls dies zutrifft, wird das Tupel (A, B) entfernt. Alle übrig gebliebenen Tupel stellen die maximal Places dar.

Y_L = { {(a), (b, e)}, {(a), (c, e)}, {(b, e), (d)}, {(c, e), (d)} }

6. Jedes Element aus Y_L gehört zu einem place p(A, B). Zusätzlich wird der Startplace i_L und der Endplace o_L erstellt.

P_L = { p{(a), (b, e)}, p{(a), (c, e)}, p{(b, e), (d)}, p{(c, e), (d)}, i_L, o_L }

7. Alle Arcs des Petri-Netzes werden erstellt. Alle Start-Transitionen aus T_I haben i_L als Input-Place und alle End-Transitionen aus T_O = {e} haben o_L als Output-Place. Alle Places p(A, B) haben A als Input-Knoten und B als Output-Knoten.

F_L = { (a, p{(a), (b, e)}), (p{(a), (b, e)}, b),(p{(a), (b, e)}, e), (a, p{(a), (c, e)}), (p{(a), (c, e)}, c), (p{(a), (c, e)}, e),(b, p{(b, e), (d)}), (e, p{(b, e), (d)}) (p{(b, e), (d)}, d), (c, p{(c, e), (d)}), (e, p{(c, e), (d)}), (p{(c, e), (d)}, d), (i_L, a), (d, o_L) }

8. Im letzten Schritt wird das Petri-Netz aus den Places, Transitionen und Arcs erzeugt.

$\alpha(L) = (P_L, T_L, F_L)$

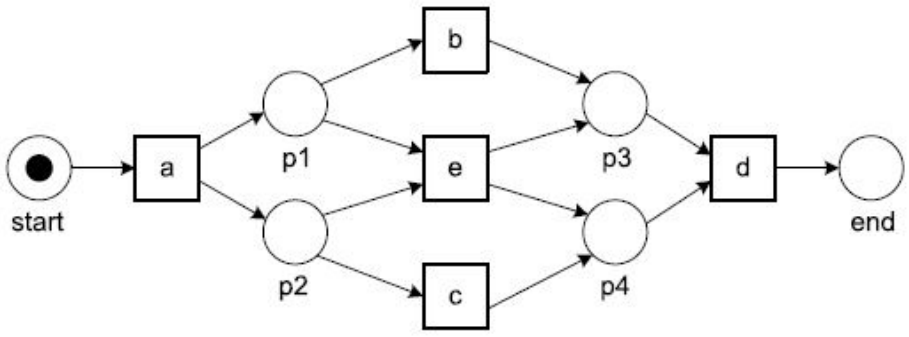

Abbildung 2.11: Prozess-Modell des Eventlogs L_1 (Aalst 2016, S.164)

2.2.3 Einschränkungen des α-Algorithmus

Schleifenkonstrukte

Der α-Algorithmus bringt einige Einschränkungen mit sich. Eine dieser Einschränkungen ist der Umgang mit Schleifenkonstrukten der Länge kleiner als 3, wobei die Länge einer Schleife der Anzahl von verschiedenen Transitionen entspricht.

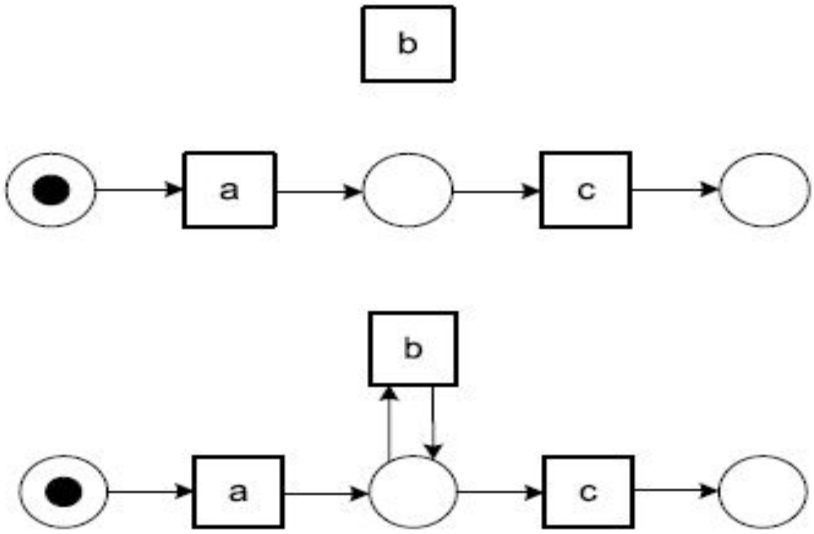

Abbildung 2.12: Einschränkungen bei Schleifen der Größe 1 (Aalst 2016, S.175)

In Abbildung 2.12 kann das Problem des α-Algorithmus bezüglich Schleifen der Größe 1 beobachtet werden. Das obere Prozess-Modell wurde von dem α-Algorithmus erzeugt, das untere Prozess-Modell hätte erzeugt werden müssen. Das Problem liegt darin, dass die Schleifen einer Länge 1 im Eventlog die Form (..., b, b, b, b...) annehmen. Wie im unteren Prozess-Modell zu sehen ist, müsste ein Place erzeugt werden, welcher b als Input und zugleich auch als Output besitzt. Dies würde für den Footprint b $>$ b und b $<$ b bedeuten, was nicht möglich ist (vgl. de Medeiros u. a. 2003, S. 396).

Um dieses Problem zu beseitigen, muss Pre- und Post-Processing verwendet werden. Im Pre-Processing werden die Schleifen der Größe 1 identifiziert und Vorgänger- und Nachfolger-Transitionen gemerkt. Anschließend werden diese Schleifen aus dem jeweiligen Case entfernt, der Case jedoch wird nicht entfernt. Danach wird der α-Algorithmus

angewendet. Im Post-Processing wird die Schleife mit den vorher gemerkten Nachbarn verbunden (vgl.de Medeiros u. a. 2003, S.400-402).

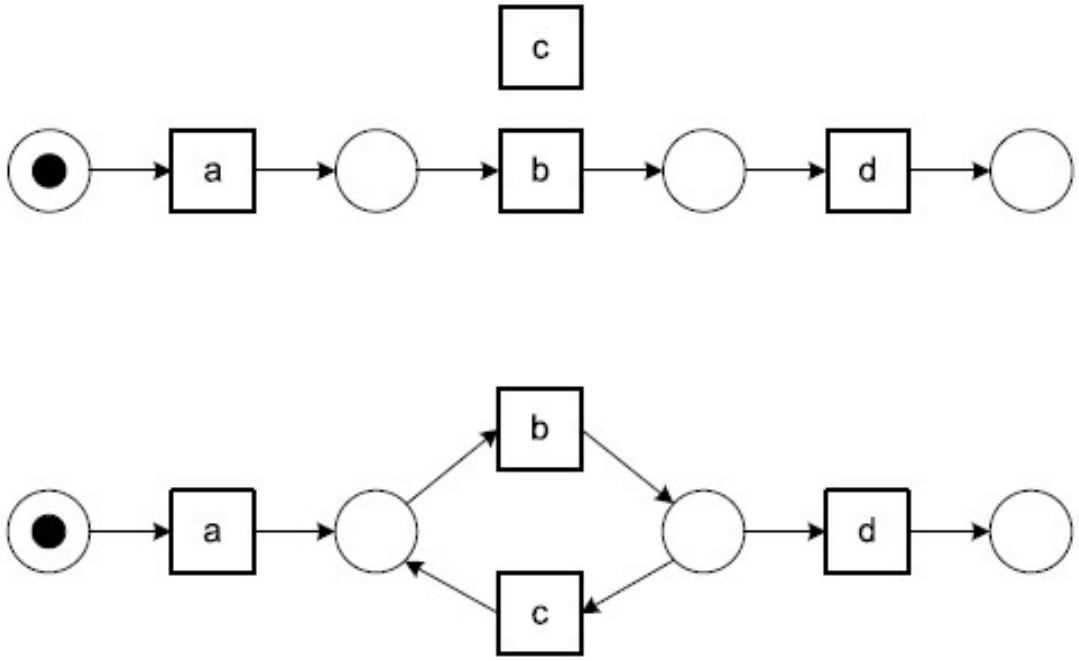

Abbildung 2.13: Einschränkungen bei Schleifen der Größe 2 (Aalst 2016, S.176)

In Abbildung 2.12 ist das Problem des α-Algorithmus bezüglich Schleifen der Größe 2 zu sehen. Im Eventlog würde ein solches Konstrukt die Form (..., a, b, a, ...) annehmen. Dies führt dazu, dass zwischen a und b eine Parallelitätsrelation (a||b), anstelle einer Kausalitätsrelation (a -> b, b -> a) angenommen wird (vgl. de Medeiros u. a. 2003, S. 396) . Die Lösung für dieses Problem besteht darin, dass die Ordnungsrelationen neu definiert werden. Zwei Aktivitäten a und b stehen nur dann in Parallelitätsrelation zueinander, wenn im Eventlog kein Trace mit der Form (..., a, b, a, ...) existiert (vgl. de Medeiros u. a. 2003, S. 402-403).

Noise

Noise beschreibt die Begebenheit, dass das Eventlog seltenes und unregelmäßig auftretendes Verhalten beinhaltet, welches nicht dem typischen Verhalten des Prozesses entspricht

(vgl. Aalst 2016, S. 185). Der α-Algorithmus bezieht die Häufigkeit der im Log enthaltenen Traces nicht mit in das Ergebnis ein, d.h., Noise wird von dem α-Algorithmus nicht erkannt. Das Herausfiltern von Noise ist sehr wichtig für Process-Mining und verschiedene Process-Discovery-Ansätze, wie z.B. Heurisic-Mining oder Genetic-Mining haben sich darin spezialisiert (vgl. Aalst 2016, S.186).

Incompleteness

Wenn ein Eventlog zu wenig Events beinhaltet, um die zugrundeliegenden Kontrollfluss-Strukturen zu identifizieren, handelt es sich um Incompleteness. Um die Bedeutung von Incompleteness bzw. Completeness zu verdeutlichen, kann man sich einen Prozess bestehend aus 10 Aktivitäten, welche parallel ausgeführt werden können, und ein Eventlog mit 10.000 Cases vorstellen.

Die Anzahl von möglichen Traces würde 10! = 3.628.000 betragen, es ist also unmöglich, dass die 10.000 im Eventlog enthaltenen Cases das komplette Verhalten des Prozesses abbilden können. Selbst bei 3.628.000 Cases im Eventlog, wäre es sehr unwahrscheinlich, dass alle mögliche Varianten vertreten sind. Um dieses Problem zu vermeiden, hat der α-Algorithmus eine relative schwache Auffassung von Completeness. Der α-Algorithmus nutzt eine local-Completeness, die auf der $>_{\mathrm{L}}$ Relation basiert, d.h., wenn es zwei Aktivitäten a und b gibt, und a kann direkt gefolgt werden von b, sollte dies mindestens einmal im Eventlog beobachtet werden können. Die Anzahl der benötigten Cases reduziert sich dadurch auf 10x9 = 90 (vgl. Aalst 2016, S.187).

2.2.4 Qualitätskriterien eines Prozess-Modells

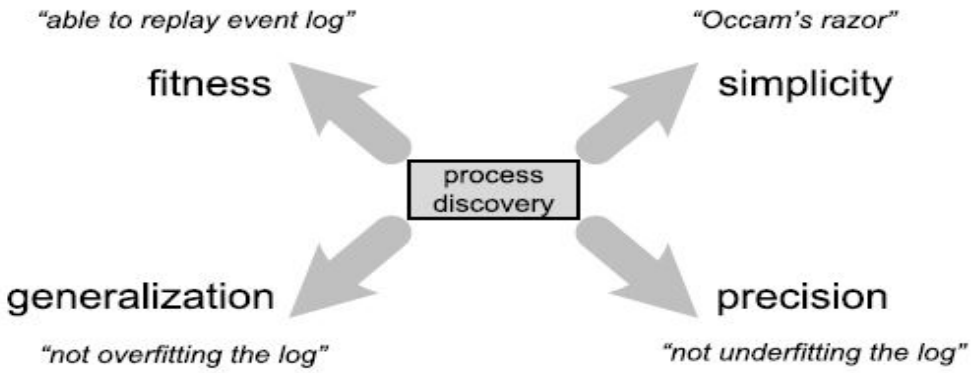

Abbildung 2.14: Balance der vier Qualitätsdimensionen (Aalst 2016, S.189)

Um die Qualität eines entdeckten Prozess-Modells zu bewerten, existieren folgende vier Kriterien (vgl. Aalst 2016, S.166):

- Fitness: Das entdeckte Prozess-Modell sollte das im Eventlog beobachte Verhalten erlauben.
- Precision: Das entdeckte Prozess-Modell sollte kein Verhalten zulassen, welches vollständig von dem beobachteten Verhalten abweicht.
- Generalization: Das entdeckte Prozess-Modell sollte das im Eventlog beobachtete Beispiel-Verhalten generalisieren.
- Simplicity: Das entdeckte Prozess-Modell sollte so einfach sein wie möglich.

Die vier genannten Dimensionen können alle quantifiziert werden (vgl. Rozinat und Aalst 2008, S. 8-24). Die Quantifizierung des Kriteriums Fitness wird in Abschnitt 2.4 genauer beschrieben.

In Abbildung 2.15 können vier unterschiedliche Prozess-Modelle beobachtet werden, die alle anhand desselben Eventlogs erstellt wurden. Diese vier Modelle unterscheiden sich durch ihre unterschiedlichen Ausprägungen der Qualitätskriterien.

N_4 zeigt ein sogenanntes Enumerating-Modell. Bei dieser Art werden alle möglichen Traces als Sequenz dargestellt. Das daraus resultierende Prozess-Modell hat zwar eine perfekte Fitness und Precision, aber es ist sehr umständlich und *overfitting*, d.h. es hat sich zu sehr an das beobachte Verhalten im Eventlog angepasst.

Bei N_3 handelt es sich um ein Flower-Modell. Alle Transitionen werden über einen Place miteinander verbunden. Das Prozess-Modell weist zwar eine perfekte Fitness auf, hat die Fähigkeit zu generalisieren und ist einfach zu verstehen, aber es lässt Verhalten zu, welches stark vom Eventlog abweicht. Ein Modell, was eine geringe Precision besitzt, ist *underfitting*.

N_2 ist ein Prozess-Modell, welches genau einem Trace aus dem Eventlog entspricht. Dieses Prozess-Modell ist einfach und hat eine hohe Precision, dafür ist die Fitness und die Fähigkeit zu generalisieren sehr gering.

N_1 stellt das beste Prozess-Modell für das abgebildete Eventlog dar, weil alle vier Qualitätskriterien erfüllt sind. Die herausfordernste Aufgabe bezüglich der Qualitätskriterien ist, eine perfekte Balance zwischen *overfitting* und *underfitting* zu finden (vgl. Aalst 2016, S.190). Das Prozess-Modell sollte gleichzeitig nicht nur das beobachte Verhalten im Eventlog zulassen, sondern auch ähnliches Verhalten, welches nicht im Eventlog verzeichnet ist. Die Schwierigkeit dabei ist, zu bestimmen, welches Verhalten ähnlich ist oder komplett abweicht.

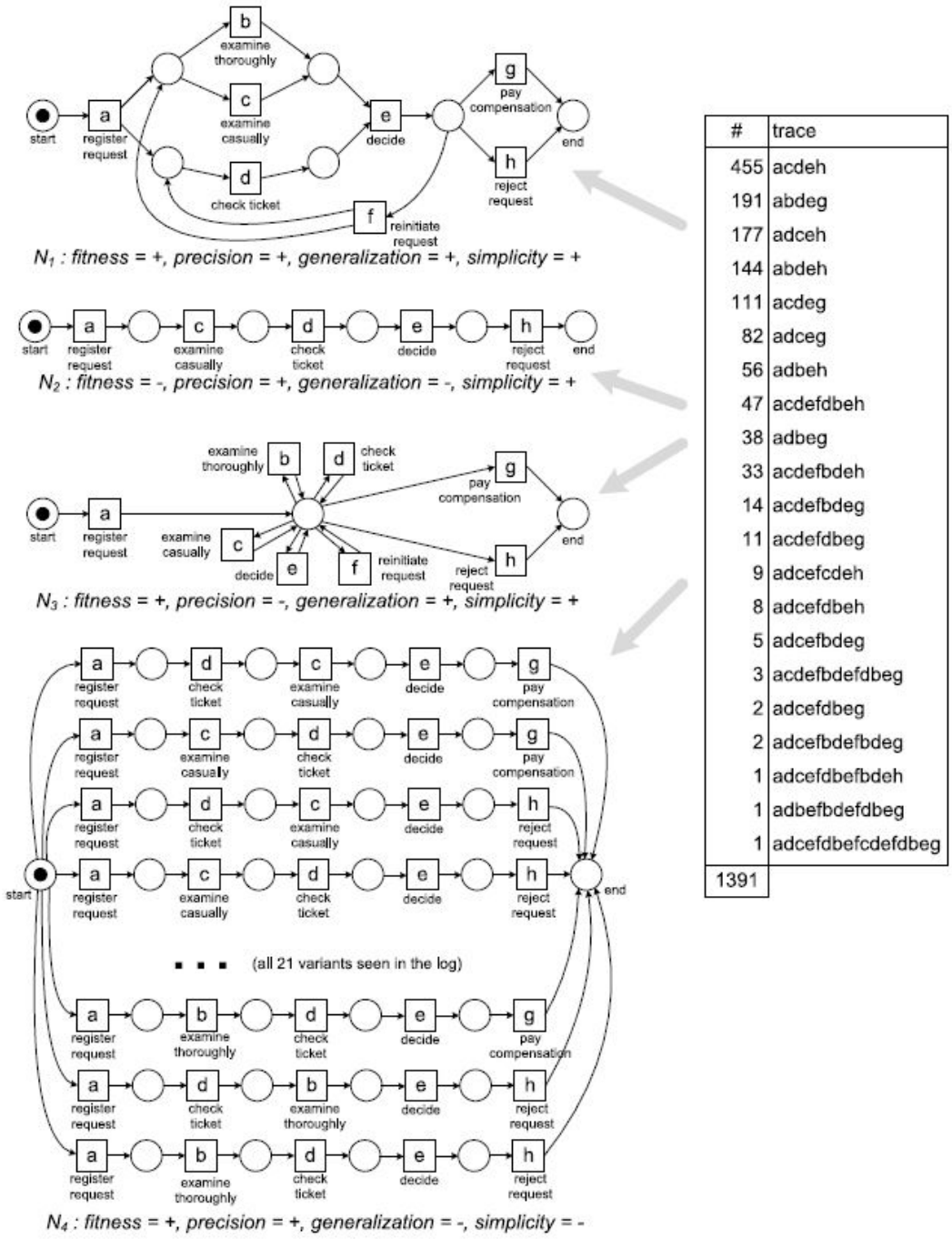

Abbildung 2.15: Erklärung der vier Qualitätskriterien anhand eines Eventlogs (Aalst 2016, S.191)

2.3 Process-Discovery: Fortgeschrittene Algorithmen

Der α-Algorithmus hat die Grundgedanken hinter Process-Discovery veranschaulicht, aber dieser simple Algorithmus kann nicht alle Probleme handhaben, die das Thema Process-Discovery mit sich bringt. Um Process-Discovery erfolgreich in der Praxis anzuwenden, müssen die Algorithmen in der Lage sein mit Noise und Incompleteness umzugehen. In diesem Kapitel werden die Eigenschaften von Process-Discovery-Algorithmen erläutert, die Einfluss auf die Auswahl des konkreten Algorithmus haben und fortgeschrittenere Algorithmen vorgestellt. Ziel dieses Kapitels ist dabei nicht, die einzelnen Algorithmen im Detail zu erklären, sondern einen Überblick zu geben und ein Bewusstsein dafür zu schaffen, wann welcher Algorithmus geeignet ist.

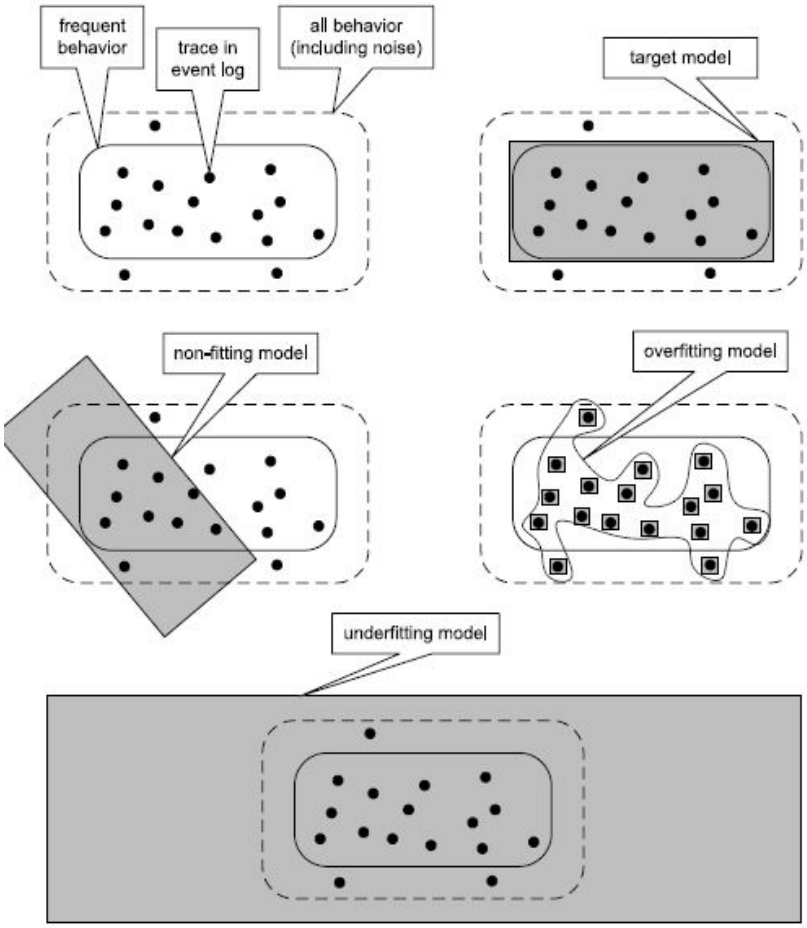

Abbildung 2.16: Überblick der Herausforderungen für Process-Discovery-Techniken (Aalst 2016, S.196)

Abbildung 2.16 fasst die Probleme zusammen, die im Kontext des α-Algorithmus erwähnt worden sind. Jeder schwarze Punkt repräsentiert dabei einen Trace, der zu einem oder mehreren Cases im Eventlog gehört. Ein Eventlog beinhaltet neben häufig auftretendem Verhalten auch unregelmäßig auftretendes Verhalten. Es kann zwar interessant sein, diese Ausreißer zu untersuchen, aber bei der Konstruktion des Prozess-Modells führt dieses außergewöhnliche Verhalten zu einer erhöhten Komplexität. Um zwischen gewöhnlichem und ungewöhnlichem Verhalten zu unterscheiden, kann die 80/20-Regel angewendet, d.h., dass das normale Verhalten den 80% der am häufigsten auftretenden Traces entspricht (vgl. Aalst 2016, S. 195-196).

2.3.1 Eigenschaften von Process-Discovery-Algorithmen

Repräsentationsmöglichkeiten

Die erste und wahrscheinlich wichtigste Eigenschaft von Process-Discovery-Algorithmen ist die Repräsentationsmöglichkeit, d.h. die Klasse von Prozess-Modellen, die entdeckt werden können. Der α-Algorithmus ist zum Beispiel nur in der Lage ein Petri-Netz zu entdecken, in dem jede Transition eine eindeutige und sichtbare Beschreibung hat. Die Auswahl einer bestimmten Repräsentationsmöglichkeit kann Einschränkungen mit sich bringen, wie z.B. (vgl. Aalst 2016, S.197-198):

- Unfähigkeit Parallelität darzustellen.
- Unfähigkeit mit Schleifenkonstrukten umzugehen.
- Unfähigkeit stille Aktionen darzustellen. Eine stille Aktion könnte z.B. das Überspringen von einer Aktivität sein.
- Unfähigkeit doppelte Aktionen darzustellen.
- Unfähigkeit OR-Split/Joins zu modellieren.

Umgang mit Noise

Außergewöhnliches Verhalten sollte nicht mit in das entdeckte Prozess-Modell aufgenommen werden. Dieses Verhalten kann beispielsweise durch Pre-Processing entfernt werden (vgl. Aalst 2016, S.198).

Annahmen zur Completeness

Viele Process-Discovery-Algorithmen machen implizite oder explizite Annahmen zur Completeness. Der α-Algorithmus hat eine schwache Annahme zur Completeness, was dazu führt, dass die entdeckten Prozess-Modelle *underfitting* sind. Gegensätzlich führen starken Annahmen zur Completeness dazu, dass die entdeckten Prozess-Modelle *overfitting* sind. Ein Beispiel für eine solche starke Annahme wäre, dass ein Eventlog alle möglichen Traces enthalten muss (vgl. Aalst 2016, S. 199).

2.3.2 Fortgeschrittene Algorithmen

Heuristic-Mining

Der Unterschied von Heuristic-Mining zu dem α-Algorithmus besteht neben einer anderen Repräsentation darin, dass die Häufigkeit von Events mit in die Konstruktion des Prozess-Modells einbezogen wird. Der Grundgedanke dahinter ist, dass außergewöhnliches Verhalten nicht in das Modell mit eingebunden wird (vgl. Aalst 2016, S. 201). Folglich ist Heuristic-Mining weniger empfindlich gegenüber Noise (vgl. Weijters und Aalst 2003, S. 5).

Um die Basiskonzepte von Heuristic-Mining darzustellen, wird folgendes Eventlog verwendet:

L=[(a, e)^{5}, (a, b, c, e)^{10}, (a, c, b, e)^{10}, (a, c, e)^{1}(a, d, e)^{10}, (a, d, d, e)^{2}, (a, d, d, d, e)^{1}]

Zunächst wird die Häufigkeit benötigt, wie oft ein Event von einem anderen gefolgt wird. Dies ergibt in diesem Beispiel folgende Matrix:

$\lvert >_L \rvert$	*a*	*b*	*c*	*d*	*e*
a	0	11	11	13	5
b	0	0	10	0	11
c	0	10	0	0	11
d	0	0	0	4	13
e	0	0	0	0	0

Abbildung 2.17: Häufigkeit der Relation a > b (Aalst 2016, S. 203)

Auf Basis dieser Häufigkeitsmatrix kann die sogenannte Abhängigkeits-Kennzahl berechnet werden. Diese Kennzahl kann einen Wert zwischen -1 und 1 annehmen.
Wenn |a -> b| nahe 1 ist, dann existiert ein starker positiver Zusammenhang zwischen a und b, d.h. a wird oft gefolgt von b.

$\lvert\Rightarrow_L\rvert$	a	b	c	d	e
a	$\frac{0}{0+1}=0$	$\frac{11-0}{11+0+1}=0.92$	$\frac{11-0}{11+0+1}=0.92$	$\frac{13-0}{13+0+1}=0.93$	$\frac{5-0}{5+0+1}=0.83$
b	$\frac{0-11}{0+11+1}=-0.92$	$\frac{0}{0+1}=0$	$\frac{10-10}{10+10+1}=0$	$\frac{0-0}{0+0+1}=0$	$\frac{11-0}{11+0+1}=0.92$
c	$\frac{0-11}{0+11+1}=-0.92$	$\frac{10-10}{10+10+1}=0$	$\frac{0}{0+1}=0$	$\frac{0-0}{0+0+1}=0$	$\frac{11-0}{11+0+1}=0.92$
d	$\frac{0-13}{0+13+1}=-0.93$	$\frac{0-0}{0+0+1}=0$	$\frac{0-0}{0+0+1}=0$	$\frac{4}{4+1}=0.80$	$\frac{13-0}{13+0+1}=0.93$
e	$\frac{0-5}{0+5+1}=-0.83$	$\frac{0-11}{0+11+1}=-0.92$	$\frac{0-11}{0+11+1}=-0.92$	$\frac{0-13}{0+13+1}=-0.93$	$\frac{0}{0+1}=0$

Abbildung 2.18: Abhängigkeits-Kennzahlen (Aalst 2016, S. 204)

Anhand dieser Häufigkeiten und Abhängigkeits-Kennzahlen können Schranken definiert werden. Relationen zweier Aktivitäten, die unterhalb dieser Schranke liegen, werden bei der Konstruktion nicht beachtet. Der Graph für die Häufigkeits-Schranke 2 und die Abhängigkeits-Schranke 0.7 sieht folgendermaßen aus:

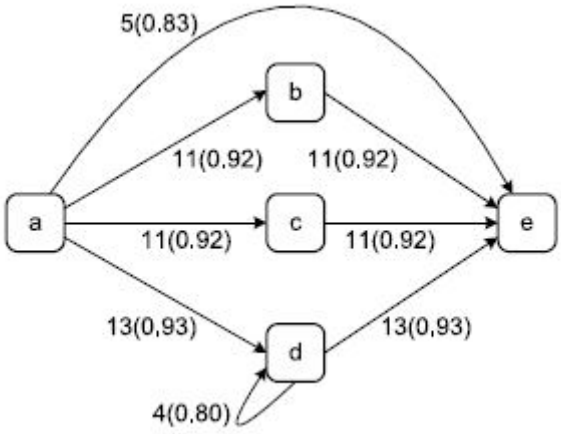

Abbildung 2.19: Abhängigkeits-Graph für Häufigkeits-Schranke 2 und Abhängigkeits-Schranke 0.7 (Aalst 2016, S. 204)

Durch Anpassen der Schranken auf 5 bzw. 0.9, ändert sich der Graph wie folgt:

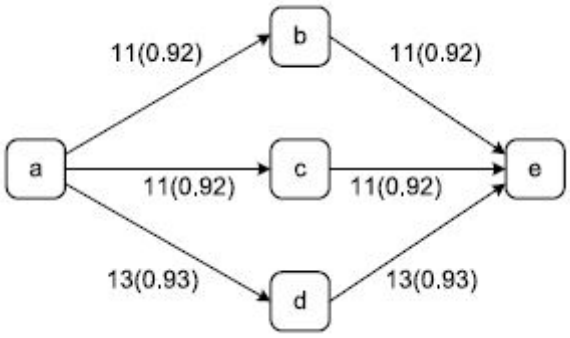

Abbildung 2.20: Abhängigkeits-Graph für Häufigkeits-Schranke 5 und Abhängigkeits-Schranke 0.9 (Aalst 2016, S.205)

Genetic-Mining

Der α-Algorithmus und die Techniken für Heuristic-Mining erzeugen Prozess-Modelle in einer direkten und deterministischen Weise. Dem gegenüber steht Genetic-Mining, welches eine evolutionäre Herangehensweise mit einem iterativen Prozess verwendet, um den Prozess der natürlichen Evolution zu imitieren (vgl. de Medeiros u.a. 2007, S. 251). Eine solche Herangehensweise arbeitet nicht-deterministisch und es ist zufällig, welche neuen Prozess-Modelle entdeckt werden (vgl. Aalst 2016, S. 207). Im Folgenden werden die vier Hauptbestandteile des Genetic-Minings beschrieben (vgl. Aalst 2016, S.208-209):

- Initialisierung: In diesem Schritt wird eine Startbevölkerung kreiert, die aus hunderten bis zu tausenden Individuen bestehen kann. In diesem Kontext ist ein Individuum ein Prozess-Modell, welches zufällig basierend auf den Aktivitätsnamen im Eventlog erzeugt wird. Die erzeugten Prozess-Modelle werden sich in der ersten Generation stark von dem im Eventlog beobachteten Verhalten unterscheiden, aber per Zufall könnten Teile von ihnen dem Eventlog entsprechen.
- Auswahl: Für die Auswahl der Prozess-Modelle wird ein Maßstab benötigt. In diesem Fall wird von Fitness gesprochen, die aber nicht mit der Fitness im vorigen Kapitel verwechselt werden darf. Im Kontext von Genetic-Mining umfasst Fitness alle vier Qualitätskriterien, um die besten Prozess-Modelle auszuwählen. Beispielsweise könnten die besten 1% der Prozess-Modelle für die nächste Generation ausgewählt

werden, wohingegen der Rest in einer Art *Turnier* antreten muss. Die besten Individuen in diesem *Turnier* werden für die Reproduktion ausgewählt und als *Parents* bezeichnet.

- Reproduktion: Die ausgewählten Individuen werden genutzt, um mit Hilfe von zwei genetischen Operatoren neue *Children* zu erschaffen. Bei Crossover werden zwei Individuen genommen und aus ihnen zwei neue Prozess-Modelle erzeugt. Die daraus resultierenden Prozess-Modelle werden per Mutation verändert, indem zufällig eine kausale Abhängigkeit hinzugefügt oder gelöscht wird.
- Terminierung: Die vorigen Schritte werden solange durchgeführt, bis in einer Generation ein Prozess-Modell entstanden ist, welches über einen gewissen Fitness-Grad verfügt, welcher im Vorfeld festgelegt werden muss. Dies kann in der Praxis sehr lange dauern, weswegen der Prozess des Genetic-Minings oft abgebrochen wird, wenn eine gewisse Anzahl an Generationen erreicht oder beispielsweise seit 10 Generationen kein besseres Prozess-Modell erzeugt worden ist.

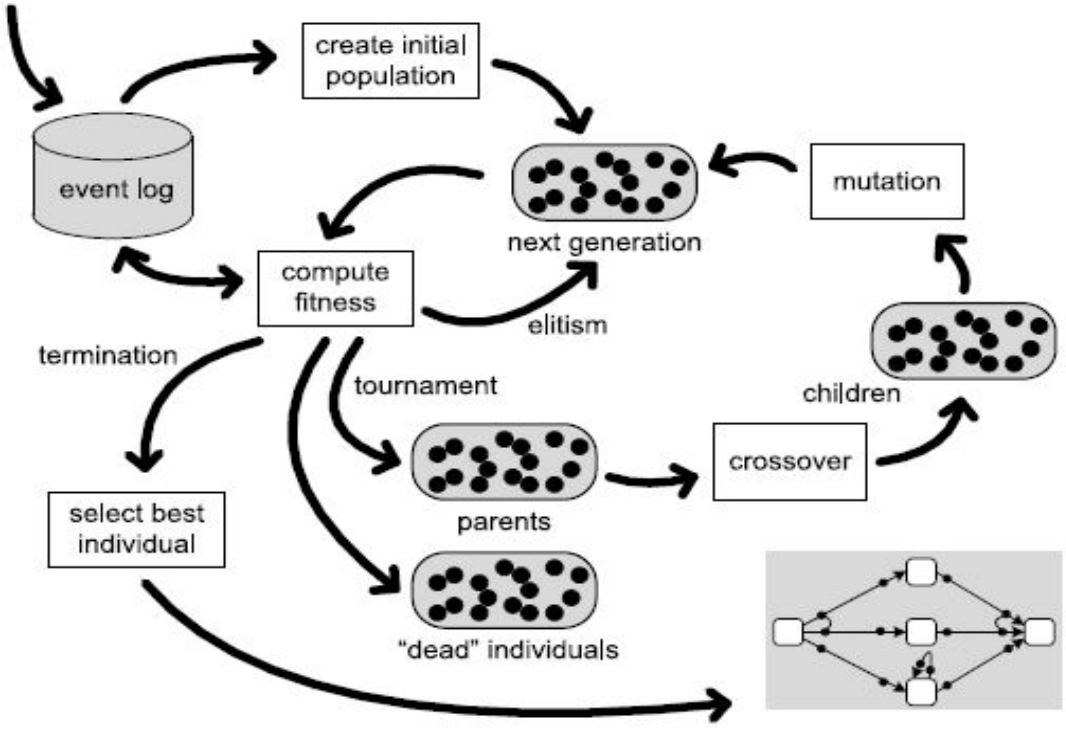

Abbildung 2.21: Überblick Genetic-Mining (Aalst 2016, S.208)

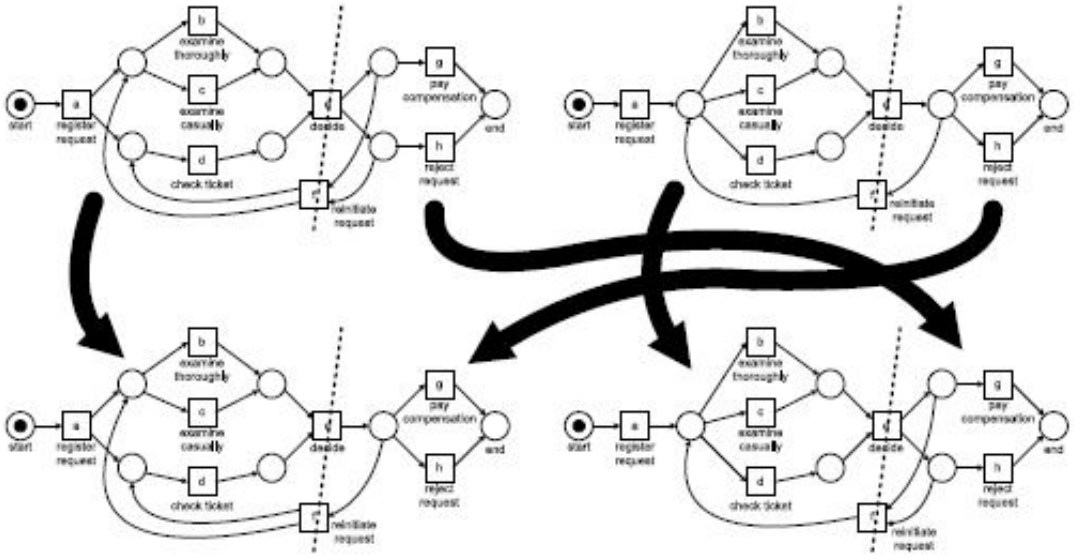

Abbildung 2.22: Zwei Parents-Modelle (oben) und zwei Children-Modelle (unten) erzeugt durch Crossover. Die gestrichelten Linien zeigen den Ort des Crossovers an (Aalst 2016, S.211)

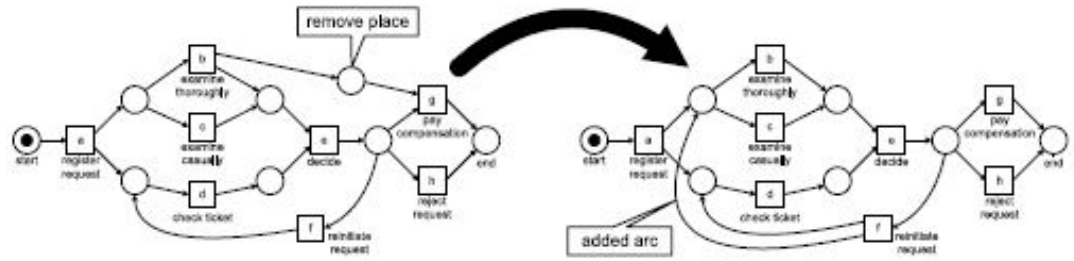

Abbildung 2.23: Mutation: Ein Place wurde entfernt und ein Arc hinzugefügt (Aalst 2016, S.211)

Inductive-Mining

Inductive-Mining basiert auf dem *Teile-und-Herrsche*-Prinzip. Die Grundidee hinter Inductive-Mining ist das Aufteilen des Eventlogs in mehrere kleine Eventlogs. Inductive-Mining ist zurzeit eine der führenden Process-Discovery-Ansätze aufgrund seiner Flexibilität, formalen Korrektheit und Skalierbarkeit (vgl. Aalst 2016, S. 222).

Ausgangspunkt für Inductive-Mining ist ein *Directly-Follows-Graph*. In Abbildung 2.24 ist dieser Graph für das Beispiel-Eventlog L_1 = [(a, b, c, d)^3, (a, c, b, d)^2, (a, e, d)] zu sehen. Inductive-Mining arbeitet mit vier Operationen, auch als Cut bezeichnet, um den Graphen und dazugehörig das Eventlog aufzuteilen.

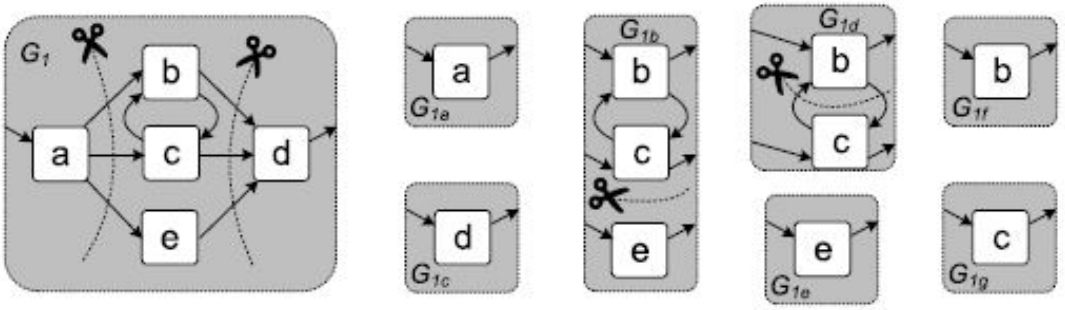

Abbildung 2.24: Directly-Follows-Graph (Aalst 2016, S.223)

Die vier möglichen Operationen bzw. Cuts lauten wie folgt:

1. Exclusive-Choice-Cut
2. Sequence-Cut
3. Parallel-Cut
4. Redo-Loop-Cut

Das Ergebnis von Inductive-Mining liegt in der Form eines Process-Trees vor. Um dieses Ergebnis zu erhalten, wird versucht die Operationen in der oben genannten Reihenfolge auf den Directly-Follows-Graph auszuführen.

Ist beispielsweise kein Exclusive-Choice-Cut möglich, wird als nächster der Sequence-Cut versucht. Ist der Sequence-Cut nicht möglich, wird versucht den Parallel-Cut auszuführen. Sollte auch dieser nicht möglich sein, wird zuletzt der Redo-Loop-Cut ausprobiert. Sobald ein Cut möglich ist und das Eventlog aufgeteilt werden konnte, wird der Vorgang wiederholt. Dieser iterative Prozess wird solange fortgeführt, bis nur atomare Eventlogs bestehend aus einer Aktivität vorhanden sind.

In diesem Beispiel wird zuerst ein Sequence-Cut durchgeführt und man erhält die Logs L_{1a}, L_{1b} und L_{1c}. L_{1a} und L_{1c} sind atomar und müssen nicht weiter betrachtet werden. Auf L_{1b} wird danach ein Exclusive-Choice-Cut ausgeführt, das Resultat sind die Logs

L_{1d} und L_{1e}. L_{1e} ist atomar und muss nicht weiterbearbeitet werden. Zuletzt wird auf L_{1d} ein Parallel-Cut angewendet und erzeugt dadurch die Logs L_{1f} und L_{1g}. Da nur noch atomare Eventlogs vorhanden sind, terminiert der Algorithmus. Der dabei aufgebaute Process-Tree kann in Abbildung 2.25 betrachtet werden.

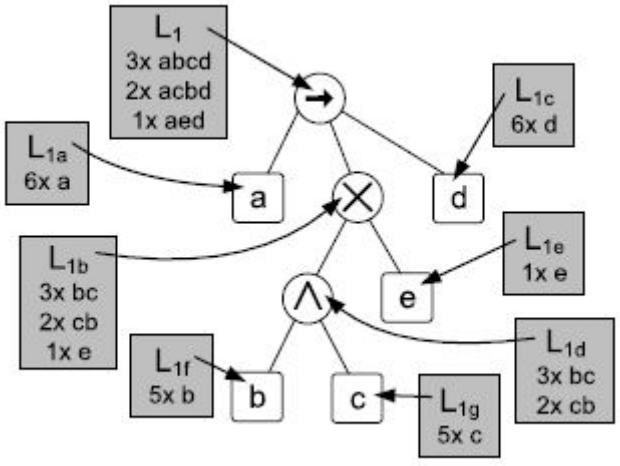

Abbildung 2.25: Process-Tree für L_1 (Aalst 2016, S.224)

2.4 Process-Conformance

Bei Process-Discovery wurde aus einem Eventlog ein Prozess-Modell erzeugt, Process-Conformance hingegen nimmt als Parameter ein Eventlog und ein Prozess-Modell entgegen und liefert als Ergebnis eine Kennzahl zurück, welche die Überstimmung von Eventlog und Prozess-Modell beschreibt. Das Ziel von Process-Conformance besteht darin sowohl Gemeinsamkeiten als auch Unterschiede des modellierten und beobachteten Verhaltens festzustellen.

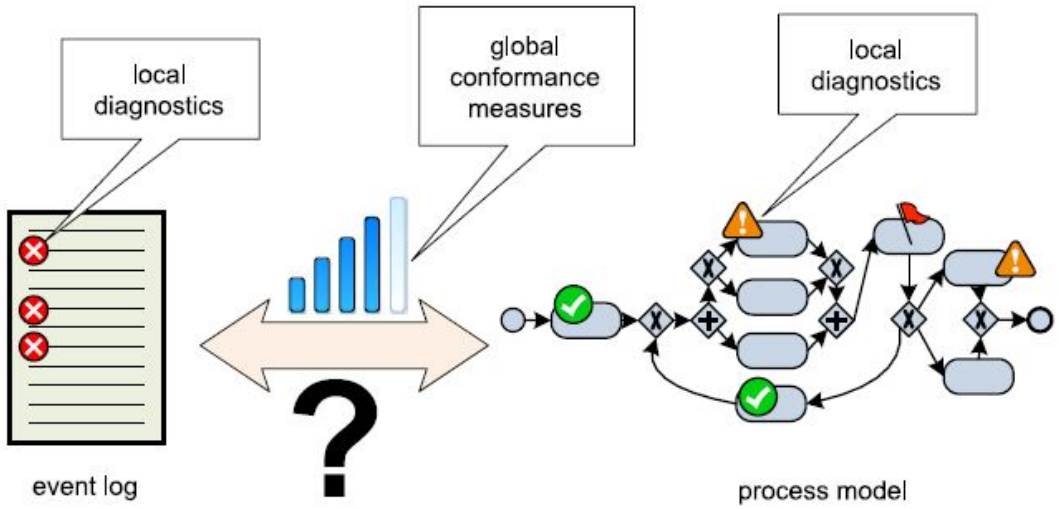

Abbildung 2.26: Conformance-Checking (Aalst 2016, S.243)

Abbildung 2.26 zeigt den Grundgedanken von Conformance-Checking. Die Analyse von Eventlog und Prozess-Modell liefert globale Übereinstimmungs-Kennzahlen (z.B. 85% der Cases im Eventlog können von dem Prozess-Modell durchgespielt werden) und lokale Diagnosen (z.B. Aktivität x wurde 15-mal ausgeführt, obwohl dies laut dem Prozess-Modell nicht möglich ist). Sobald Abweichungen festgestellt worden sind, müssen diese aus zwei Blickwinkeln betrachtet werden (vgl. Aalst 2016, S.244).

1. Das Prozess-Modell ist falsch und spiegelt nicht die Realität wider. In diesem Fall sollte das Prozess-Modell verbessert werden.

2. Die Cases weichen vom Prozess-Modell ab. Folglich sollten Methoden entwickelt werden, die zur besseren Kontrolle des Prozesses führen und abweichende Handlungen verhindern.

Des Weiteren sollte man sich bewusst machen, dass nicht jede Abweichung als negativ zu bewerten ist. Beispielweise könnte ein Arzt einem Patienten das Leben retten, indem er gerade nicht nach Protokoll handelt. Abweichungen dieser Art sollten in Zukunft auch zugelassen und in das Modell mit aufgenommen werden.

Im Folgenden werden drei verschiedene Techniken des Conformance-Checking vorgestellt. Als Grundlage für die Erläuterungen dient folgendes Eventlog, welches insgesamt 1391 Cases beinhaltet:

Frequency	Reference	Trace
455	σ_1	$\langle a, c, d, e, h \rangle$
191	σ_2	$\langle a, b, d, e, g \rangle$
177	σ_3	$\langle a, d, c, e, h \rangle$
144	σ_4	$\langle a, b, d, e, h \rangle$
111	σ_5	$\langle a, c, d, e, g \rangle$
82	σ_6	$\langle a, d, c, e, g \rangle$
56	σ_7	$\langle a, d, b, e, h \rangle$
47	σ_8	$\langle a, c, d, e, f, d, b, e, h \rangle$
38	σ_9	$\langle a, d, b, e, g \rangle$
33	σ_{10}	$\langle a, c, d, e, f, b, d, e, h \rangle$
14	σ_{11}	$\langle a, c, d, e, f, b, d, e, g \rangle$
11	σ_{12}	$\langle a, c, d, e, f, d, b, e, g \rangle$
9	σ_{13}	$\langle a, d, c, e, f, c, d, e, h \rangle$
8	σ_{14}	$\langle a, d, c, e, f, d, b, e, h \rangle$
5	σ_{15}	$\langle a, d, c, e, f, b, d, e, g \rangle$
3	σ_{16}	$\langle a, c, d, e, f, b, d, e, f, d, b, e, g \rangle$
2	σ_{17}	$\langle a, d, c, e, f, d, b, e, g \rangle$
2	σ_{18}	$\langle a, d, c, e, f, b, d, e, f, b, d, e, g \rangle$
1	σ_{19}	$\langle a, d, c, e, f, d, b, e, f, b, d, e, h \rangle$
1	σ_{20}	$\langle a, d, b, e, f, b, d, e, f, d, b, e, g \rangle$
1	σ_{21}	$\langle a, d, c, e, f, d, b, e, f, c, d, e, f, d, b, e, g \rangle$

Abbildung 2.27: Beispiel Eventlog (Aalst 2016, S.247)

2.4.1 Token-Replay

In Abschnitt 2.3.3 wurden die vier Qualitätskriterien eines Prozess-Modells vorgestellt: Fitness, Precision, Generalization und Simplicity. Mit Hilfe der Techniken des Conformance-Checkings kann das Kriterium Fitness quantifiziert werden.

Eine Möglichkeit der Quantifizierung wäre, die von einem Prozess-Modell nachspielbaren Cases in Verhältnis zu der Gesamtanzahl der Cases zu setzen. Ein Prozess-Modell N_1, welches alle Cases nachspielen könnte, hätte die Fitness $\frac{1391}{1391} = 1$. Diese Quantifizierung auf Case-Ebene bringt aber ein wesentliches Problem mit sich. Sobald nur ein Event in einem Case nicht nachgespielt werden kann, wird der ganze Case als nicht-nachspielbar bewertet. Besteht ein Case beispielsweise aus 1.000 Events und 1 Event davon kann nicht nachgespielt werden, würde der komplette Case als nicht-nachspielbar bewertet werden, obwohl eine Übereinstimmung von fast 100% erreicht wurde.

Die Quantifizierung auf Case-Ebene ist folglich zu strikt und lässt keine differenziertere Bewertung der Cases zu. In der Hierarchie von Process-Mining befindet sich unter der Case-Ebene nur noch die Event-Ebene (vgl. Aalst 2016, S. 147). Folglich wird eine Quantifizierung auf Event-Ebene benötigt, wie z.B. das Token-Replay, um eine differenzierte Bewertung zuzulassen.

Das Token-Replay berechnet die Fitness von Eventlog und Prozess-Modell mit Hilfe der produzierten (p), konsumierten (c), fehlenden (m) und verbleibenden (r) Tokens. Die Formel für die Berechnung lautet wie folgt:

$$fitness(\sigma, N) = \frac{1}{2}\left(1 - \frac{m}{c}\right) + \frac{1}{2}\left(1 - \frac{r}{p}\right)$$

Abbildung 2.28: Formel zur Berechnung der Fitness im Kontext von Token-Replay (Aalst 2016, S.250)

Damit eine Transition aktiv wird, benötigt sie die gleiche Anzahl von Tokens, wie sie eingehende Arcs bzw. Input-Places hat. Nachdem eine Transition aktiv war, erzeugt sie für jeden ausgehenden Arc bzw. Output-Place ein Token.

Token-Replay kann als eine Art Spiel angesehen werden, bei dem man die Spielsteine/Tokens durch das Prozess-Modell schieben muss. Sollte das Ende erreicht werden, ohne dass Spielsteine übrig bleiben, hat man das Spiel gewonnen (d.h. eine Fitness von

1). Das Spiel gilt dann als verloren, wenn Spielsteine übrigbleiben oder zu wenig Spielsteine vorhanden sind, um zum Ende zu gelangen (d.h. eine Fitness kleiner als 1, aber größer gleich 0).

Die Idee des Token-Replays wird anhand der Abbildung 2.29 verdeutlicht. Bei diesem Beispiel soll ein Prozess-Modell N den Trace σ = (a, d, c, e, h) nachspielen.
Initial ist p = c = m = r = 0 und alle Places haben keine Tokens.

1. Es wird von der Umgebung ein Token für den Place p_{start} produziert, dementsprechend erhöhen wir p um 1. p = 1, c = m = r = 0.

2. Das erste Event aus dem Trace (a) kann nachgespielt werden. Die Transition a konsumiert das Token und produziert zugleich 1 Token für Place p_1. Wir erhöhen c und p um 1. p = 2, c = 1, m = r = 0.

3. Als nächstes wird versucht das Event d nachzuspielen, was nicht möglich ist, da die Transition nicht aktiviert werden kann. Um Transition d trotzdem zu aktivieren, damit mit dem Token-Replay fortgefahren werden kann, wird ein Token dem Place p_2 hinzugefügt. Transition d kann nun aktiv werden, konsumiert das Token von $Place_{p2}$ und erzeugt ein Token für Place p_3. Wir erhöhen m, p und c um 1. p = 3, c = 2, m = 1, r = 0.

4. Anschließend wird c nachgespielt. Die Transition c wird aktiv, konsumiert das Token von Place p_1 und produziert ein Token für Place p_2. Wir erhöhen p und c um 1. p = 4, c = 3, m = 1 und r = 0.

5. Darauf wird e nachgespielt. Die Transition e wird aktiv, konsumiert das Token von Place p_3 und erzeugt ein Token für Place p_4. Wir erhöhen p und c um 1. p = 5, c = 4, m = 1 und r = 0.

6. Danach wird h nachgespielt. Die Transtion h wird aktiv, konsumiert das Token von Place p_4 und erzeugt ein Token für Place p_{end}. Wir erhöhen p und c um 1. p = 6, c = 5, m = 1 und r = 0.

7. Zuletzt konsumiert die Umgebung das Token von Place p_{end}. Der Trace wurde zu Ende durchgespielt, aber in Place p_2 ist ein Token übrig geblieben. Wir erhöhen c und r um 1. Das Endergebnis lautet p = 6, c = 6, m = 1 und r = 1.

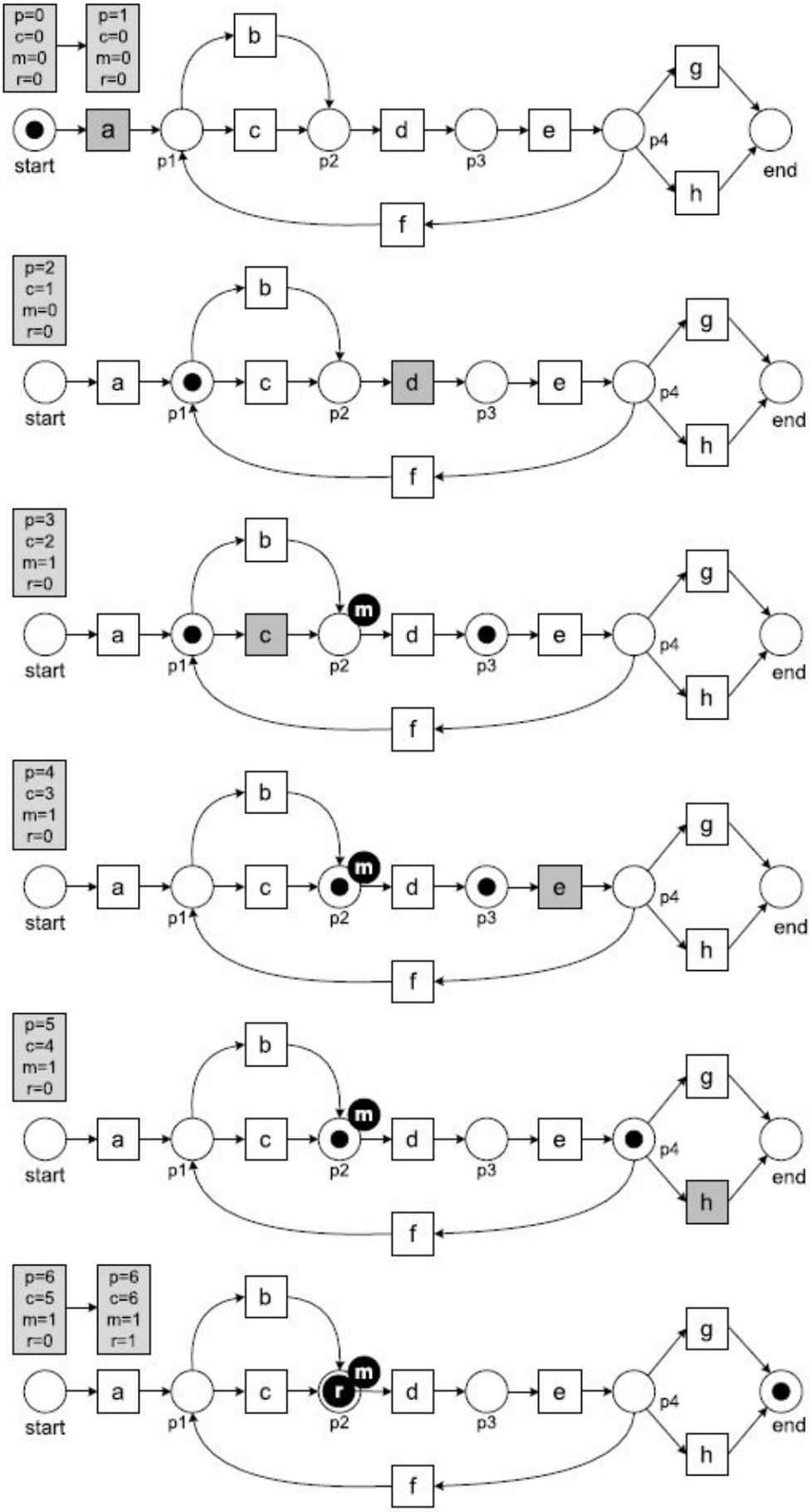

Abbildung 2.29: Token-Replay (Aalst 2016, S.249)

Mit Hilfe der oben genannten Formel kann die Fitness berechnet werden:
fitness(σ, N) = $\frac{1}{2}$(1-$\frac{1}{6}$)+$\frac{1}{2}$(1-$\frac{1}{6}$) = 0.8333

Im Vergleich dazu würde das in Abbildung 2.29 verwendete Prozess-Modell nur insgesamt 948 Cases komplett durchspielen können und bei einer Fitness-Berechnung auf Case-Ebene den Wert $\frac{948}{1391}$ = 0.6815 erhalten.

In dem oben gezeigten Beispiel wurde die Fitness basierend auf einem Trace berechnet. Um die Fitness für ein Eventlog zu berechnen, muss jeder einzelne Case nachgespielt werden. Die Einzelwerte werden am Ende für die Variablen p, c, m und r zusammengezählt und anschließend die Formel angewendet.

2.4.2 Alignment

Eine weitere Technik des Conformance-Checking ist Alignment. Dieses Verfahren hat den großen Vorteil, dass es für jegliche Art von Prozess-Modellen geeignet ist. Token-Replay hingegen ist auf Petri-Netze beschränkt und jede andere Art von Prozess-Modell müsste zuvor in ein Petri-Netz umgewandelt werden. Ein weiterer Vorteil von Alignment ist, dass versucht wird, das beobachte Verhalten auf das Prozess-Modell zu übertragen, auch wenn der jeweilige Case nicht komplett nachspielbar ist (vgl. Aalst 2016, S. 256).

Der Grundgedanke von Alignment ist, dass ein Case und ein möglicher Pfad eines Prozess-Modells übereinandergelegt und abgeglichen werden. Wenn das Prozess-Modell nicht in der Lage ist den Case vollständig nachzuspielen, sollte der Pfad dem Case so gut wie möglich ähneln. Als ein optimales Alignment wird die bestmögliche Übereinstimmung bei gegebenen Case und Prozess-Modell bezeichnet.

$$\gamma_1 = \begin{array}{|c|c|c|c|c|} a & d & b & e & h \\ \hline a & d & b & e & h \end{array}$$

Abbildung 2.30: Alignment bei 100% Übereinstimmung (Aalst 2016, S.257)

Die obere Reihe entspricht dem zu untersuchenden Case, die untere Reihe entspricht dem Pfad des Prozess-Modells. In diesem Fall kann der Case vollständig nachgespielt werden und es liegen keinerlei Abweichungen vor.

Sollten Case und Prozess-Modell voneinander abweichen, würde sich folgendes optimales Alignment ergeben:

$$\gamma_{5,2a} = \begin{array}{|c|c|c|c|c|} a & b & \gg & d & f \\ a & b & c & d & f \\ t1 & t2 & t3 & t5 & t8 \end{array} \qquad \gamma_{5,2b} = \begin{array}{|c|c|c|c|c|} a & \gg & b & d & f \\ a & c & b & d & f \\ t1 & t3 & t2 & t5 & t8 \end{array}$$

Abbildung 2.31: Optimales Alignment bei Abweichungen (Aalst 2016, S.257)

Bei diesem Beispiel existieren zwei optimale Alignments. Die Doppelpfeile (») signalisieren Abweichungen zwischen Case und Pfad des Prozess-Modells. Ein Doppelpfeil in der oberen Reihe bedeutet, dass in dem Case eine Aktion, die hätte ausgeführt werden müssen, übersprungen worden ist. Ein Doppelpfeil in der unteren Reihe gibt an, dass in dem Case eine Aktion ausgeführt wurde, die laut Prozess-Modell an dieser Stelle nicht erlaubt ist. Auf dieses Beispiel bezogen bedeutet das, dass der Case die Aktion c überspringt, obwohl diese laut Prozess-Modell an zweiter oder dritter Stelle hätte ausgeführt müssen (vgl. Aalst 2016, S. 257)

Den Misaligments können Kosten zugewiesen werden, wodurch man die Prägnanz von bestimmten Aktionen ausdrücken kann. Beispielsweise könnte die Aktion c den Wert 3 bekommen, wohingegen anderen Aktionen der Wert 1 zugewiesen wird. Zur Vereinfachung wird bei der Erläuterung jeder Aktion der Wert 1 zugewiesen, d.h. die Kosten der Misaligments entsprechen der Anzahl der Misaligments (vgl. Aalst 2016, S. 260).

Um mit Hilfe von Alignments das Kriterium Fitness zu quantifizieren, kann folgende Formel verwendet werden:

$$fitness(\sigma, N) = 1 - \frac{\delta(\lambda^N_{opt}(\sigma))}{\delta(\lambda^N_{worst}(\sigma))}$$

Abbildung 2.32: Formel zur Berechnung der Fitness im Kontext von Alignment (Aalst 2016, S.260)

Zu Berechnung der Fitness wird neben den Kosten des optimalen Alignments auch die Kosten des suboptimalsten Alignments benötigt. Das suboptimalste Alignment für den oben verwendeten Case sieht folgendermaßen aus:

$$\gamma_{5,2w} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline a & b & d & f & \gg & \gg & \gg & \gg \\ \hline \gg & \gg & \gg & \gg & a & c & d & f \\ & & & & t1 & t3 & t4 & t8 \\ \hline \end{array}$$

Abbildung 2.33: Suboptimalstes Alignment(Aalst 2016, S.260)

Das suboptimalste Alignment entsteht, wenn der Case und der kürzeste Pfad durch ein Prozess-Modell komplett versetzt angeordnet werden. Sofern allen Misalignments die gleichen Kosten zugewiesen werden, setzen sich die Kosten für das suboptimalste Alignment aus der Anzahl der Events im Case und der Länge des kürzesten Pfades zusammen, was in diesem Fall 8 wäre.

Dadurch würde sich sich die Fitness 1- $\frac{1}{8} = 0.875$ ergeben. Um die Fitness für ein ganzes Eventlog zu berechnen, müssen die Einzelwerte für die Kosten von optimalen und suboptimalsten Alignments zusammengefasst werden.

2.4.3 Footprint-Vergleich

In Abschnitt 2.3.2. wurde im Rahmen des α-Algorithmus vier Ordnungsrelationen eingeführt, mit deren Hilfe die Beziehungen zwischen den einzelnen Aktivitäten eines Eventlogs beschrieben werden konnten. Um diese vier Ordnungsrelationen kompakt und übersichtlich darzustellen, wurde eine sogenannte Footprint-Matrix verwendet. Die Footprint-Matrix charakterisiert das Verhalten eines Eventlogs und kann folglich als Ausgangspunkt für Conformance-Checking herangezogen werden (vgl. Aalst 2016, S. 243).

Bisher wurde erläutert, wie die Footprint-Matrix für ein Eventlog erstellt werden kann. Da Conformance-Checking ein Eventlog mit einem Prozess-Modell vergleicht, ist es notwendig auch eine Footprint-Matrix für ein Prozess-Modell erzeugen zu können. Dafür muss zuerst aus dem Prozess-Modell ein vollständiges Eventlog generiert werden, d.h. sofern eine Aktivität von einer anderen Aktivität gefolgt werden kann, muss dies mindestens einmal im Eventlog beobachtet werden können.

Die Idee des Footprint-Vergleichs wird in folgender Abbildung deutlich:

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
a	#	→	→	→	#	#	#	#
b	←	#	#	‖	→	←	#	#
c	←	#	#	‖	→	←	#	#
d	←	‖	‖	#	→	←	#	#
e	#	←	←	←	#	→	→	→
f	#	→	→	→	←	#	#	#
g	#	#	#	#	←	#	#	#
h	#	#	#	#	←	#	#	#

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
a	#	→	→	#	#	#	#	#
b	←	#	#	→	#	←	#	#
c	←	#	#	→	#	←	#	#
d	#	←	←	#	→	#	#	#
e	#	#	#	←	#	→	→	→
f	#	→	→	#	←	#	#	#
g	#	#	#	#	←	#	#	#
h	#	#	#	#	←	#	#	#

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
a				→: #				
b				‖ :→	→: #			
c				‖ :→	→: #			
d	←: #	‖ :←	‖ :←			←: #		
e		←: #	←: #					
f				→: #				
g								
h								

Abbildung 2.34: Vergleich auf Basis zweier Footprint-Matrizen (Aalst 2016, S.264)

Die obere Footprint-Matrix ist aus dem Eventlog erzeugt worden, welches als Grundlage für die Erläuterungen des Conformance-Checking diente. Die mittlere Footprint-Matrix basiert auf dem Prozess-Modell, welches in Abschnitt 2.5.1 für das Token-Replay verwendet wurde. Die untere Footprint-Matrix zeigt die Unterschiede der beiden Matrizen. In

der Unterschied-Footprint-Matrix haben nur die Zellen einen Wert, bei denen die beiden zu vergleichenden Matrizen eine Abweichung vorweisen. Der Wert beinhaltet die angenommenen Relationen beider Matrizen, beispielsweise bedeutet der Wert in der ersten Zeile, dass Matrix 1 eine *Directly-Follows*-Relation (->) zwischen den Aktivitäten a und d annimmt, wohingegen Matrix 2 davon ausgeht, dass keine direkte Verbindung zwischen diesen Aktivitäten existiert (#).

Die Fitness bei dem Footprint-Vergleich kann wie folgt berechnet werden:
Fitness = 1 - $\frac{Anzahl\ abweichender\ Zellen}{Gesamtanzahl\ der\ Zellen}$
In diesem Fall beträgt die Fitness den Wert 1- $\frac{12}{64} = 0.8125$

Die gezeigten Techniken wurden genutzt, um die Übereinstimmung zwischen einem Eventlog und einem ganzen Prozess-Modell zu überprüfen. Darüber hinaus ist es möglich, gezielt auf spezifische Vorgaben, sogenannte Business-Rules, zu untersuchen. Beispielsweise könnte untersucht werden, ob Aktivität a immer von Aktivität b gefolgt wird oder ob Aktivität a und b nicht von der gleichen Person ausgeführt werden, damit das *4-Augen-Prinzip* nicht verletzt wird (vgl. Aalst 2016, S. 266).

2.5 Process-Enhancement

Während sich Process-Discovery ausschließlich um die Kontrollflussperspektive kümmert, versucht Prozess-Enhancement weitere Informationen aus den Eventlogs zu verwenden, um andere Perspektiven auf den zugrundeliegenden Prozess zu gewähren. Dieser Bereich von Prozess-Enhancement wird als Extension bezeichnet. In diesen Abschnitt werden die Möglichkeiten der Organisations-, Zeit- und Caseperspektive erläutert.

2.5.1 Organisationsperspektive

Ausgangspunkt für die Organisations-Perspektive ist das Attribute Ressource. Als Basis für die Erläuterungen dient das aus Abschnitt 2.2.2 bekannte Eventlog, welches zusätzlich um das Transaktiontyp-Attribut erweitert wurde:

Case id	Event id	Properties				
		Time	Activity	Trans	Resource	Cost
1	35654423	30-12-2010:11.02	register request	start	Pete	
	35654424	30-12-2010:11.08	register request	complete	Pete	50
	35654425	31-12-2010:10.06	examine thoroughly	start	Sue	
	35654427	31-12-2010:10.08	check ticket	start	Mike	
	35654428	31-12-2010:10.12	examine thoroughly	complete	Sue	400
	35654429	31-12-2010:10.20	check ticket	complete	Mike	100
	35654430	06-01-2011:11.18	decide	start	Sara	
	35654431	06-01-2011:11.22	decide	complete	Sara	200
	35654432	07-01-2011:14.24	reject request	start	Pete	
	35654433	07-01-2011:14.32	reject request	complete	Pete	200
2	35654483	30-12-2010:11.32	register request	start	Mike	
	35654484	30-12-2010:11.40	register request	complete	Mike	50
	35654485	30-12-2010:12.12	check ticket	start	Mike	
	35654486	30-12-2010:12.24	check ticket	complete	Mike	100
	35654487	30-12-2010:14.16	examine casually	start	Pete	
	35654488	30-12-2010:14.22	examine casually	complete	Pete	400
	35654489	05-01-2011:11.22	decide	start	Sara	
	35654490	05-01-2011:11.29	decide	complete	Sara	200
	35654491	08-01-2011:12.05	pay compensation	start	Ellen	
	35654492	08-01-2011:12.15	pay compensation	complete	Ellen	200
...	...	...	...	...	...	...

Abbildung 2.35: Erweitertes Eventlog (Aalst 2016, S.277)

Mit Hilfe der Organisationsperspektive kann mehr über die Menschen, Organisationsstrukturen (Rollen und Abteilungen), Arbeitsverteilung und Arbeitsmuster in Erfahrung gebracht werden (vgl. Aalst 2016, S. 281). Dafür empfiehlt es sich, das Eventlog auf das Ressource-Attribut zu projizieren, d.h. alle anderen Informationen werden zwecks Übersichtlichkeit ausgeblendet. Die kompakte Repräsentation des projizierten Eventlogs würde wie folgt aussehen:

Case id	Trace
1	$\langle a^{Pete}, b^{Sue}, d^{Mike}, e^{Sara}, h^{Pete} \rangle$
2	$\langle a^{Mike}, d^{Mike}, c^{Pete}, e^{Sara}, g^{Ellen} \rangle$
3	$\langle a^{Pete}, c^{Mike}, d^{Ellen}, e^{Sara}, f^{Sara}, b^{Sean}, d^{Pete}, e^{Sara}, g^{Ellen} \rangle$
4	$\langle a^{Pete}, d^{Mike}, b^{Sean}, e^{Sara}, h^{Ellen} \rangle$
5	$\langle a^{Ellen}, c^{Mike}, d^{Pete}, e^{Sara}, f^{Sara}, d^{Ellen}, c^{Mike}, e^{Sara}, f^{Sara}, b^{Sue}, d^{Pete}, e^{Sara}, h^{Mike} \rangle$
6	$\langle a^{Mike}, c^{Ellen}, d^{Mike}, e^{Sara}, g^{Mike} \rangle$
...	...

Abbildung 2.36: Kompakte Darstellung bezüglich des Attributs Ressource (Aalst 2016, S.282)

Aus dieser Darstellung kann sehr schnell abgelesen werden, wie die einzelnen Ressourcen an einem Case beteiligt sind. Zum Beispiel führt in Case 1 Pete die Aktivität a zuerst aus, danach folgt Sue mit Aktivität b, anschließend Mike mit Aktivität d, als nächstes Sara mit Aktivität e und zuletzt wieder Pete mit Aktivität h.

Auf Basis dessen kann eine Matrix erstellt werden, die zeigt, wie oft eine Ressource eine Aktivität pro Case im Durchschnitt ausgeführt hat:

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
Pete	0.3	0	0.345	0.69	0	0	0.135	0.165
Mike	0.5	0	0.575	1.15	0	0	0.225	0.275
Ellen	0.2	0	0.23	0.46	0	0	0.09	0.11
Sue	0	0.46	0	0	0	0	0	0
Sean	0	0.69	0	0	0	0	0	0
Sara	0	0	0	0	2.3	1.3	0	0

Abbildung 2.37: Ressource-Aktivitäts-Matrix (Aalst 2016, S.282)

Aus dieser Matrix können interessante Informationen extrahiert werden. Beispielsweise können aus ähnlichen Zeilen in der Matrix bestimmte Rollen geschlussfolgert werden. In diesem Fall führten Pete, Mike und Ellen die Aktivität a, c, d, g und h aus. Sue und Sean kümmern sich zu zweit um die Aktivität b. Aktivitäten e und f werden allein von Sara ausgeführt.

Darüber hinaus könnte erkannt werden, dass Mike eine sehr wertvolle Ressource ist, da er bei all seinen Aktivitäten öfter zum Einsatz kommt als Mike und Ellen. Des Weiteren könnte man erahnen, dass Sara eine Manager-Rolle inne hat, da sie als einzige die Entscheidung-Aktivität e ausführt.

Aus der kompakten Darstellung kann auch eine sogenannte *Handover-of-Work*-Matrix abgeleitet werden. Diese Matrix gibt an, wie oft im Durchschnitt eine Ressource von einer anderen Ressource pro Case gefolgt wird:

	Pete	Mike	Ellen	Sue	Sean	Sara
Pete	0.135	0.225	0.09	0.06	0.09	1.035
Mike	0.225	0.375	0.15	0.1	0.15	1.725
Ellen	0.09	0.15	0.06	0.04	0.06	0.69
Sue	0	0	0	0	0	0.46
Sean	0	0	0	0	0	0.69
Sara	0.885	1.475	0.59	0.26	0.39	1.3

Abbildung 2.38: Handover-of-Work-Matrix (Aalst 2016, S.284)

Es lässt sich erkennen, dass Sue und Sean nur mit Sara arbeiten und deswegen eine andere Rolle einnehmen als Pete, Mike und Ellen. Außerdem scheinen Mike und Sara stark miteinander verbunden zu sein. Auf Grundlage dieser Matrix lässt sich ein Social-Network erstellen (vgl. Song und Aalst 2008, S. 13-15). Ein Social-Network hat pro Ressource einen Knoten und ist mit anderen Knoten verbunden, sofern laut Matrix eine Verbindung existiert. Die Stärke dieser Verbindung kann durch eine unterschiedliche Pfeildicke ausgedrückt werden:

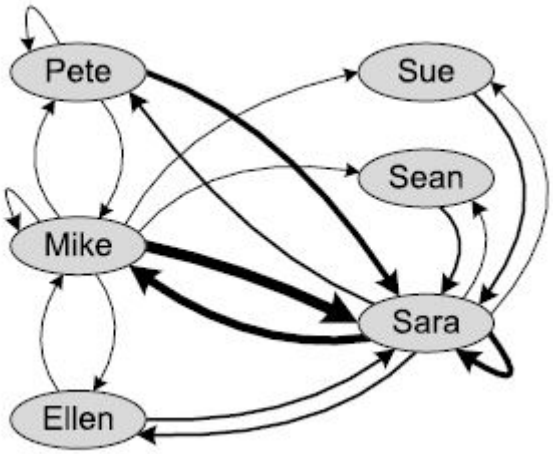

Abbildung 2.39: Social Network (Aalst 2016, S.284)

Die Ressourcen lassen sich nicht nur auf Mitarbeiterebene, sondern auch auf Rollen- oder Abteilungsebene darstellen. Aus den obigen Ausführungen lassen sich drei Rollen ableiten: Sachbearbeiter, Experte und Manager. Pete, Mike und Ellen können den Sachbearbeitern, Sue und Sean den Experten und Sara den Managern zugeordnet werden. Auf Basis dieser Zuordnung könnte ein neues Social-Network konstruiert werden.

2.5.2 Zeitperspektive

Die Zeitperspektive beschäftigt sich mit dem Zeitstempel- und Transaktions-Typ-Attribut. Diese beiden Attribute sind zwingend notwendig, um beispielweise Engpässe zu entdecken.

Analog zu der Organisationsperspektive, sollte das Eventlog auf die relevanten Attribute projiziert werden und würde wie folgt aussehen:

Case id	Trace
1	$\langle a^{12}_{start}, a^{19}_{complete}, b^{25}_{start}, d^{26}_{start}, b^{32}_{complete}, d^{33}_{complete}, e^{35}_{start}, e^{40}_{complete}, h^{50}_{start}, h^{54}_{complete}\rangle$
2	$\langle a^{17}_{start}, a^{23}_{complete}, d^{28}_{start}, c^{30}_{start}, d^{32}_{complete}, c^{38}_{complete}, e^{50}_{start}, e^{59}_{complete}, g^{70}_{start}, g^{73}_{complete}\rangle$
3	$\langle a^{25}_{start}, a^{30}_{complete}, c^{32}_{start}, c^{35}_{complete}, d^{35}_{start}, d^{40}_{complete}, e^{45}_{start}, e^{50}_{complete}, f^{50}_{start}, f^{55}_{complete}, b^{60}_{start}, d^{62}_{start}, b^{65}_{complete}, d^{67}_{complete}, e^{80}_{start}, e^{87}_{complete}, g^{90}_{start}, g^{98}_{complete}\rangle$
...	...

Abbildung 2.40: Kompakte Darstellung bezüglich der Attribute Zeitstempel- und Transaktionstyp (Aalst 2016, S.290)

Aus dieser kompakten Darstellung kann in Falle von Case 1 abgelesen werden, dass Aktivität zum Zeitpunkt 12 gestartet und zum Zeitpunkt 19 beendet wurde. Anschließend vergehen 6 Zeiteinheiten, bevor der Prozess mit Aktivität b fortgeführt wurden ist.

Auf Basis dieser Darstellung kann ein Zeitlinien-Diagramm erstellt werden, welches diese Informationen grafisch abbildet:

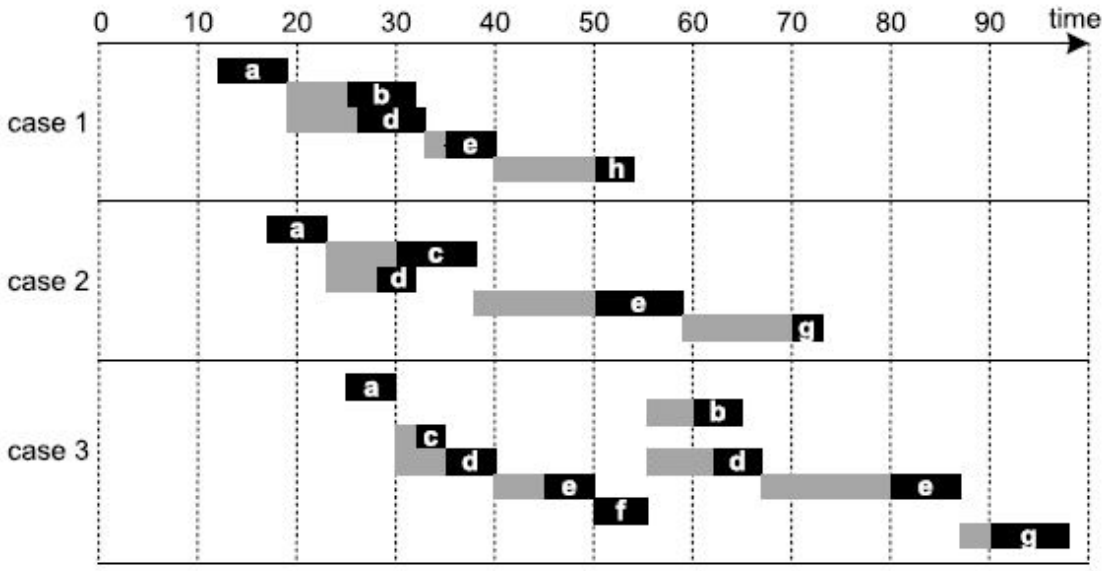

Abbildung 2.41: Zeitlinien-Diagramm (Aalst 2016, S.291)

Die schwarzen Kästchen geben den Zeitraum an, in der die Aktivität ausgeführt wurden ist, wohingegen die grauen Kästchen die Wartezeit bis zum Start der nächsten Aktivität

darstellen. Konkret in diesem Beispiel ist die Wartezeit für Aktivität e (*Decide*) auffällig, welche von der Managerin Sara durchgeführt wird.

2.5.3 Caseperspektive

Eine weitere Perspektive ist die Caseperspektive, welche nicht, wie bei den oben genannten Perspektiven, auf Attributen der Event-Ebene, sondern auf Attributen der Case-Ebene basiert. Die Attribute auf Case-Ebene sind in diesem Beispiel der Kundentyp, die Region und der Betrag.

Case id	Custid	Name	Type	Region	Amount
1	9911	Smith	gold	south	989.50
2	9915	Jones	silver	west	546.00
3	9912	Anderson	silver	north	763.20
4	9904	Thompson	silver	west	911.70
5	9911	Smith	gold	south	812.10
6	9944	Baker	silver	east	788.00
7	9944	Baker	silver	east	792.80
8	9911	Smith	gold	south	544.70
...	...	...	...	...	...

Abbildung 2.42: Case-Attribute (Aalst 2016, S.278)

In der Case-Perspektive wird nun versucht mit Hilfe von Decision-Mining dem Prozess-Modell Regeln an den Stellen hinzuzufügen, wo eine Entscheidung getroffen werden muss (XOR-Splits). Diese Regeln müssen zuerst aus Daten der Vergangenheit mit Hilfe des Decision-Tree-Algorithmus gelernt und anschließend in das Prozess-Modell integriert werden (vgl. Rozinat und Aalst 2006, S. 4-8).

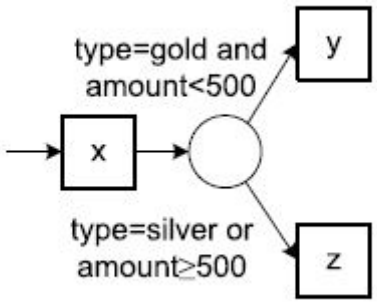

Abbildung 2.43: Regeln erzeugt durch Decision Mining (Aalst 2016, S.295)

In diesem Fall würden nach Aktivität x die Attribute Kundentyp und Betrag überprüft werden. Handelt es sich um einen Gold-Kunden oder ist der Betrag unter 500, so würde als nächste Aktivität y ausgeführt werden, bei einem Silber-Kunden oder einem Betrag über 500 würde hingegen mit der Aktivität z fortgefahren werden.

Alle genannten Perspektiven können darüber hinaus in einem einzigen Prozess-Modell vereint werden. Die Basis bildet dabei die Kontroll-Fluss-Perspektive, in der den einzelnen Transitionen oder Places des Prozess-Modell zusätzliche Informationen aus anderen Perspektiven hinzugefügt werden können.

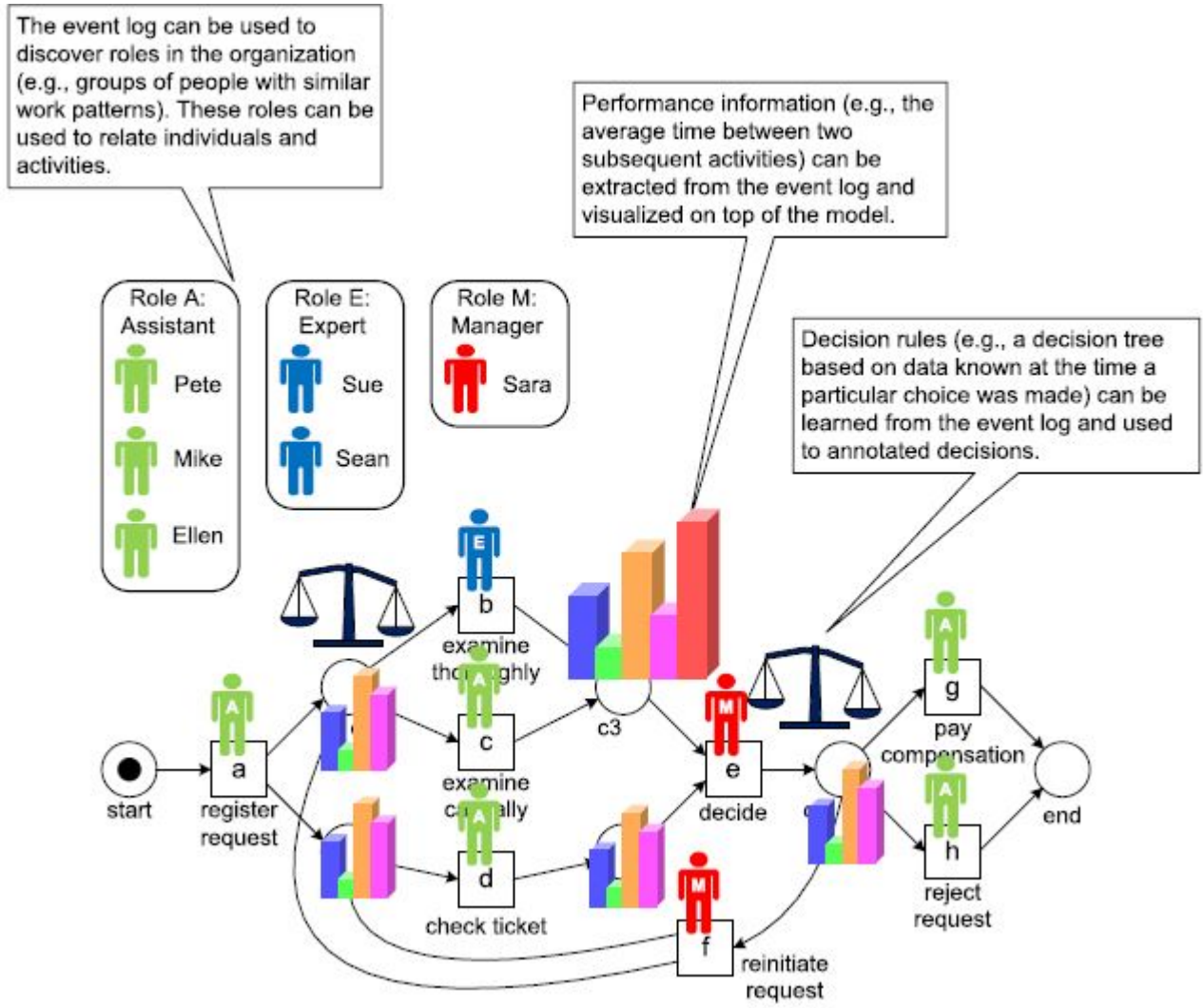

Abbildung 2.44: Integriertes Prozess-Modell (Aalst 2016, S.40)

2.6 ProM

2.6.1 Anwendung

In den vorangegangen Abschnitten wurden die theoretischen Grundlagen für Process-Mining vermittelt. Dieser Abschnitt zeigt die Anwendung von Process-Mining mit Hilfe des Tools ProM. ProM ist das führende Open-Source Process-Mining-Tool und wurde entwickelt um eine gemeinsame Basis für alle möglichen Techniken des Process-Mining, wie z.B. das Laden von Eventlogs oder das Visualisieren der Ergebnisse, zu schaffen.

Die erste Version von ProM wurde 2004 veröffentlich und beinhaltete 29 Plug-Ins. Jedes Plug-In übernimmt dabei eine abgeschlossene Aufgabe, wie z.B. die Ausführung des α-Algorithmus oder die Umwandlung eines Petri-Netzes in eine andere Darstellungsmöglichkeit. Im Laufe der Jahre sind etliche Plug-Ins dazu gekommen, sodass zurzeit über 1500 Plug-Ins verfügbar sind (vgl. Aalst 2016, S. 331-334). Aufgrund der hohen Anzahl von Plug-Ins ist es unmöglich alle Funktionalitäten von ProM zu zeigen, daher wird lediglich der Aufbau und die Arbeitsweise mit ProM beschrieben.

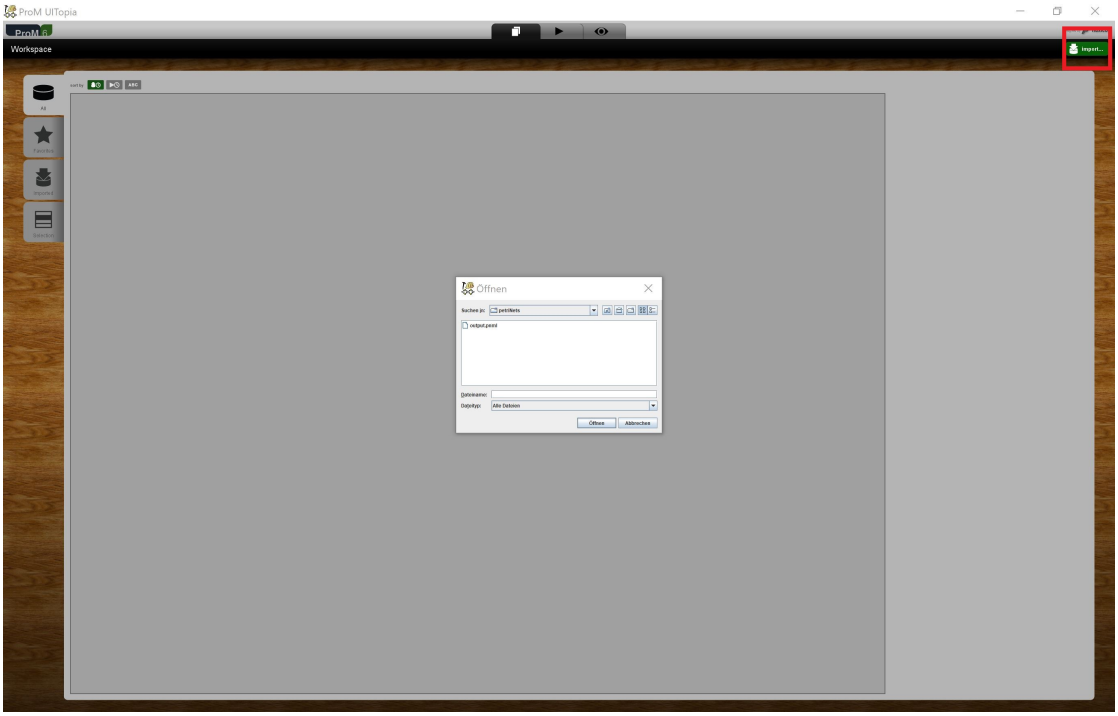

Abbildung 2.45: Import in ProM

Um die Funktionalitäten von ProM auszuführen, benötigen wir Input-Daten. Diese Daten können über den Import-Button oben rechts im Workspace importiert werden. Nach Betätigen des Import-Buttons öffnet sich ein File-Chooser-Dialog, in dem eine Datei importiert werden kann. Dabei stehen beispielsweise Eventlogs in verschiedensten Formaten (XES, XML, CSV, etc.) oder Prozess-Modelle in verschiedensten Formaten (Petri-Netze, BPMN-Modelle, etc.) zur Verfügung.

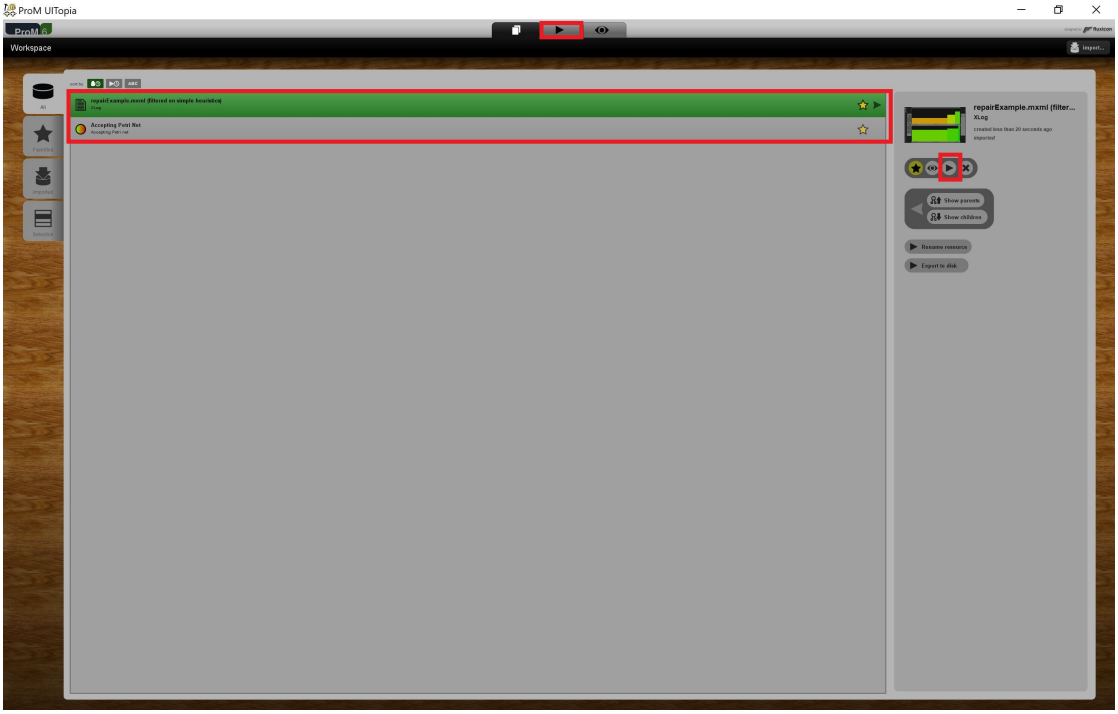

Abbildung 2.46: Workspace mit importieren Daten

Die importierten Daten sind anschließend im Workspace wiederzufinden. Wählt man einen dieser Inputs aus und betätigt einen von den beiden Aktions-Buttons (kleine rote Kästchen, Play-Symbol), gelangt man zur einer neuen Ansicht, in der eine Vielzahl von Aktionen zur Verfügung stehen, die auf dem ausgewählten Input ausgeführt werden können. Zum Beispiel könnte man den α-Algorithmus auf ein Eventlog anwenden oder die Überstimmung zwischen Eventlog und Prozess-Modell überprüfen.

Links in dieser Ansicht sind die Input-Daten aufgelistet, rechts hingegen das Ergebnis einer ausgewählten Aktion. In der Mitte sind alle verfügbaren Aktionen aufgelistet. Eine grün hinterlegte Aktion kann auf die Input-Daten angewendet werden, bei einer gelb hinterlegten Aktion fehlen notwendige Input-Daten. Wird eine gelb hinterlegte Aktion

ausgewählt, erscheinen im Input-Reiter diejenigen Daten, die hinzugefügt werden müssen, um die gewünschte Aktion auszuführen.

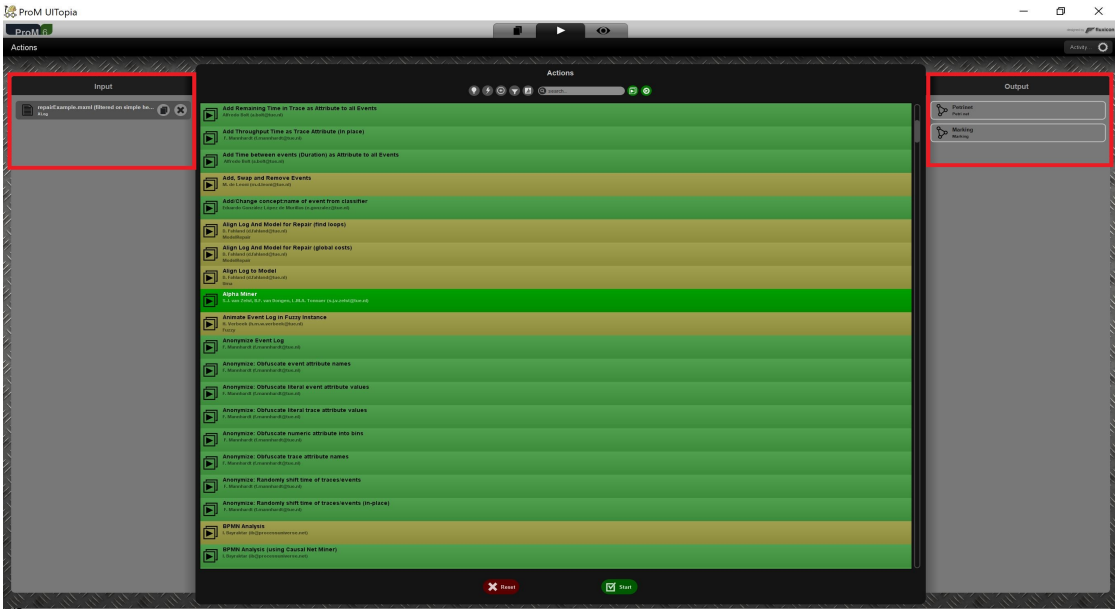

Abbildung 2.47: Aktionen in Prom

Die Ergebnisse der ausgewählten Aktionen sind daraufhin im Workspace verfügbar. Anschließend kann nach Auswählen eines Ergebnisses mit Hilfe einer der beiden View-Buttons (kleine rote Kästchen, Augen-Symbol) eine Visualisierung aufgerufen werden.

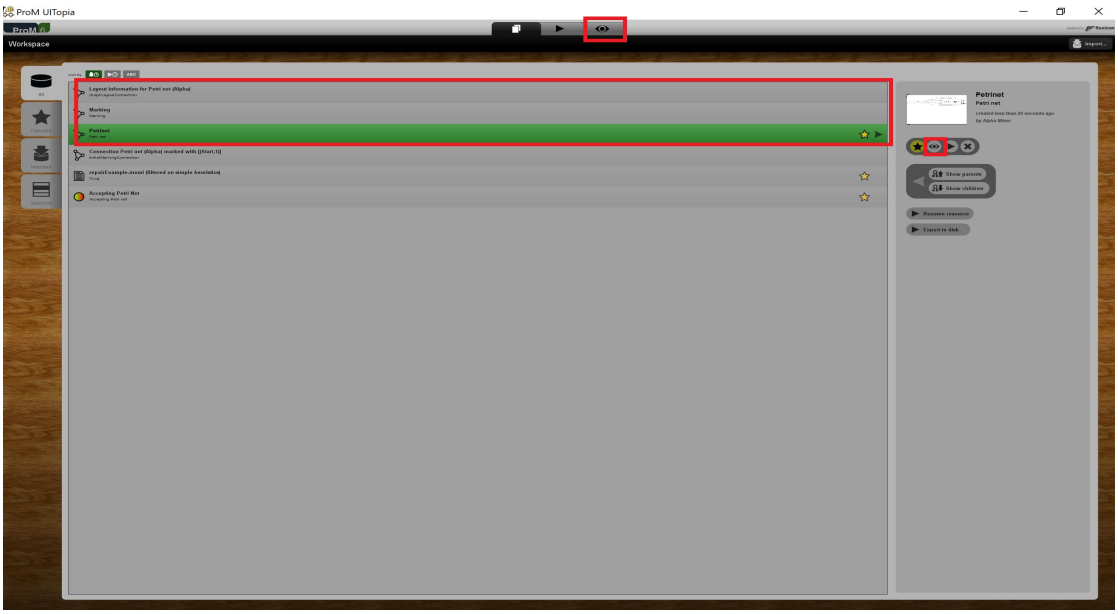

Abbildung 2.48: Workspace nach ausgeführter Aktion

Die Visualisierung ist abhängig von dem Ergebnistyp. In diesem Fall handelt es um sich ein Prozess-Modell in Form eines Petri-Netzes.

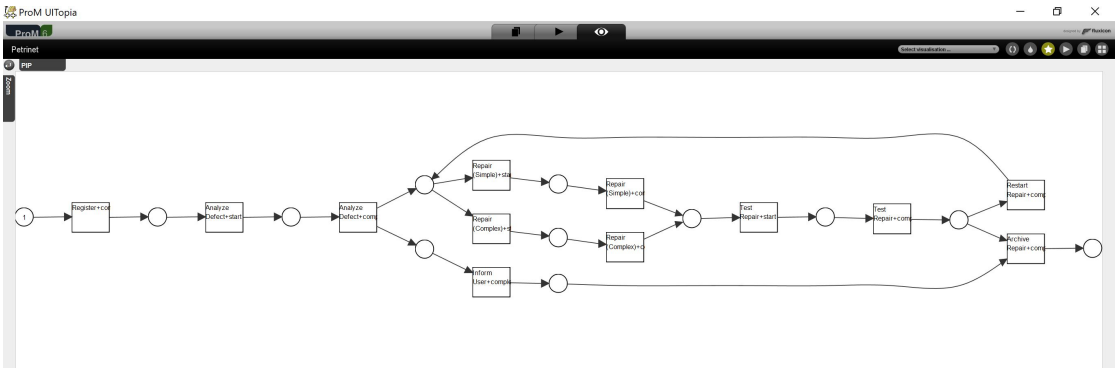

Abbildung 2.49: Visualisierung in Prom

2.6.2 Einschätzung

Das Process-Mining-Tool Prom eignet sich eher für akademische Zwecke. Diese Einschätzung beruht auf der Begebenheit, dass die grafische Oberfläche nicht ansprechend und zusätzlich auch unübersichtlich ist.

Des Weiteren wird der Anwender bei der Benutzung des Tools nicht an die Hand genommen. Nachdem ein Plug-In benutzt wurden ist, werden keine weiteren Handlungsvorschläge gegeben. Der Anwender ist selbst dafür verantwortlich, welche von den über 1500 Plug-Ins als Nächstes zum Einsatz kommen könnten.

Prom ist ein nicht-kommerzielles Open-Source Process-Mining-Tool, welches nicht auf perfekte Usability ausgelegt ist, sondern in erster Linie seinen Anwendern die vielen Möglichkeiten des Process-Mining vorstellen möchte.

Als Beispiele für Process-Mining-Tools, die auf kommerzielle Nutzung und Usability ausgelegt sind, können Celonis und Disco herangezogen werden (vgl. Aalst 2016, S. 339). Der Preis für eine bessere Usability dieser Tools liegt darin, dass sie nicht alle Möglichkeiten der 1500 Plug-Ins von Prom beinhalten.

3 Entwicklung der Bibliothek

Dieses Kapitel stellt den Entwicklungsprozess der Bibliothek zur generischen Anwendung von Process-Mining dar. Zunächst werden in Abschnitt 3.1 Anforderungen an die Bibliothek aufgestellt. Dem folgt in Abschnitt 3.2 der Entwurf, welcher mit Hilfe eines Komponenten- und eines Sequenzdiagrammes genauer erläutert wird. Abschnitt 3.3 beschreibt die Implementation dieses Entwurfes. Anschließend wird im Abschnitt 3.4 der Test dieser Bibliothek beschrieben. Im Abschnitt 3.5 wird das Deployment der Bibliothek erörtert. Zuletzt wird im Abschnitt 3.6 ein Ausblick auf die Möglichkeiten der Erweiterbarkeit dieser Bibliothek gegeben.

3.1 Requirements Engineering

Die Bibliothek zur generischen Anwendung von Process-Mining soll in Verbindung mit einer Prozessunterstützungssoftware angewendet werden. Die realen Prozesse sollen visualisiert werden, indem das Verhalten der Anwender der Prozessunterstützungssoftware analysiert wird. Die Anwender navigieren durch die Software, um eine Instanz eines Prozesses bewältigen zu können. Die Art und Weise wie ein Anwender durch die Software navigiert, kann dabei in einem Eventlog abgespeichert werden; die Prozessinstanz kann als der Case angesehen werden. Um diesen größeren Prozess zu bewältigen, sind mehrere Teilprozesse vonnöten, welche als Events interpretiert werden können. Speichert man zu den einzelnen Events zusätzlich noch die Uhrzeit ab, sind alle Grundvoraussetzungen für die Basisanwendungen von Process-Mining erfüllt:

1. Jedes Event besitzt einen Bezeichner.
2. Jedes Event muss einem Case zugeordnet sein.
3. Die Events innerhalb eines Cases müssen nach Zeitpunkt ihres Auftretens geordnet werden können.

In der Software-Entwicklung unterscheidet man grundsätzlich zwischen funktionalen und nicht-funktionalen Anforderungen. Die funktionalen Anforderungen legen fest, welche Funktionen die Software bieten muss, wohingegen die nicht-funktionalen Anforderungen beschreiben, wie gut diese Funktionen ausgeführt werden müssen.

3.1.1 Funktionale Anforderungen

Da Process-Mining aus drei großen Teilbereichen besteht, könnten auf der obersten Ebene die funktionalen Anforderungen wie folgt aussehen.

- Die Bibliothek muss die Möglichkeiten von Process-Discovery anbieten.
- Die Bibliothek muss die Möglichkeiten von Process-Conformance anbieten.
- Die Bibliothek muss die Möglichkeiten von Process-Enhancement anbieten.

Diese Anforderungen sind aber zu grobgranular und müssen daher aufgesplittet werden. Für den Teilbereich Process-Discovery lauten die funktionalen Anforderungen wie folgt:

- Die Bibliothek muss den α-Algorithmus auf ein XES-Eventlog anwenden können.
- Die Bibliothek muss den α^+-Algorithmus auf ein XES-Eventlog anwenden können.
- Die Bibliothek muss den Heuristic-Mining-Algorithmus auf ein XES-Eventlog anwenden können.
- Die Bibliothek muss den Genetic-Mining-Algorithmus auf ein XES-Eventlog anwenden können.
- Die Bibliothek muss den Inductive-Mining-Algorithmus auf ein XES-Eventlog anwenden können.

Für den Teilbereich Process-Conformance lauten die funktionalen Anforderungen wie folgt:

- Die Bibliothek muss die Token-Replay-Methode auf ein XES-Eventlog und ein Petri-Netz anwenden können.
- Die Bibliothek muss die Alignment-Methode auf ein XES-Eventlog und ein Petri-Netz anwenden können.

- Die Bibliothek muss die Footprint-Vergleich-Methode auf ein XES-Eventlog und ein Petri-Netz anwenden können.

Für den Teilbereich Process-Enhancement lauten die funktionalen Anforderungen wie folgt:

- Die Bibliothek muss aus einem XES-Eventlog mit dem Attribut Ressource eine Ressource-Aktivitäts-Matrix erstellen können.
- Die Bibliothek muss aus einem XES-Eventlog mit dem Attribut Ressource eine Handover-of-Work-Matrix erstellen können.
- Die Bibliothek muss aus einem XES-Eventlog mit dem Attribut Ressource ein Social Network erstellen können.
- Die Bibliothek muss aus einem XES-Eventlog mit den Attributen Zeitstempel und Transaktionstyp ein Zeitliniendiagramm erstellen können.
- Die Bibliothek muss aus einem XES-Eventlog mit Attributen auf Case-Ebene Entscheidungsregeln lernen können.

Neben den Funktionen aus den drei Teilbereichen von Process-Mining sind auch verschiedene Support-Funktionen von Wichtigkeit:

- Die Bibliothek muss unsortierte Eventdaten im JSON-Format entgegennehmen und ein Eventlog nach XES-Standard erstellen können.
- Die Bibliothek muss unsortierte Eventdaten im XML-Format entgegennehmen und ein Eventlog nach XES-Standard erstellen können.
- Die Bibliothek muss ein Petri-Netz in ein BPMN umwandeln können und vice versa.
- Die Bibliothek muss ein Petri-Netz in ein EPK umwandeln können und vice versa.

Die genannten Anforderungen stellen nur einen Ausschnitt der Anforderungen an die Process-Mining-Bibliothek dar. Beispielsweise würde jeder weitere Process-Discovery-Algorithmus oder jede weitere Darstellungsmöglichkeit eines Prozess-Modells eine neue funktionale Anforderung darstellen.

3.1.2 Nicht-funktionale Anforderungen

Der Standard ISO 25010 ist die internationale Norm für Qualitätskriterien von Software und kann als Basis für die nicht-funktionalen Anforderungen herangezogen werden. Insgesamt wird zwischen 8 verschiedenen Qualitätskriterien unterschieden (vgl. ISO25010 2011):

- Funktionalität
- Performance
- Kompatibilität
- Benutzbarkeit
- Zuverlässigkeit
- Sicherheit
- Wartbarkeit
- Portierbarkeit

Diese Qualitätskriterien können dabei genauer spezifiziert werden:

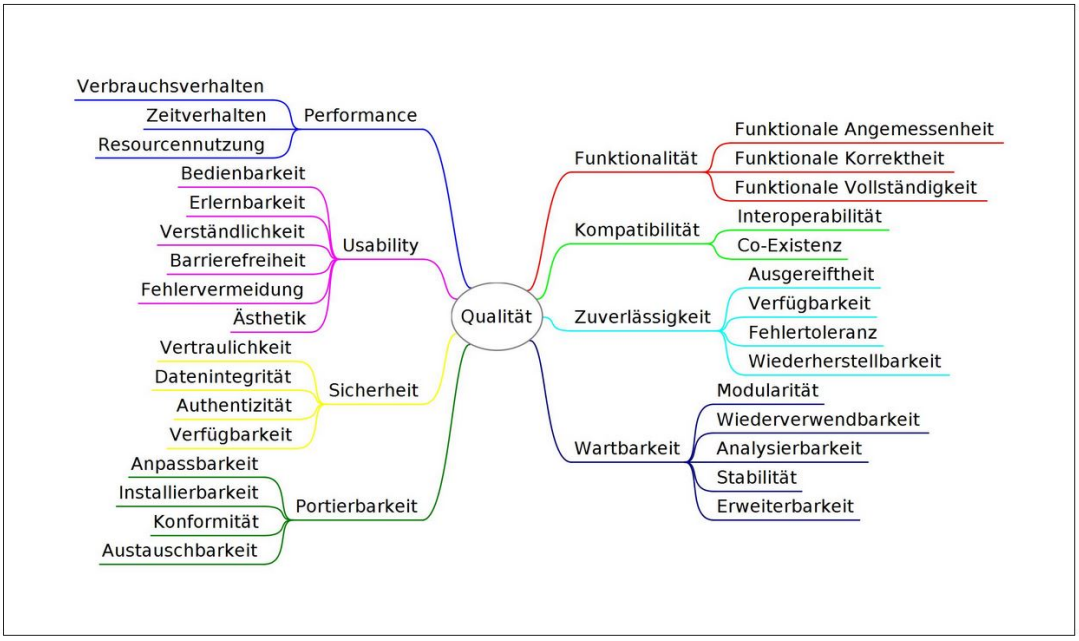

Abbildung 3.1: IS0 25010: Qualitätskriterien einer Software (Kops 2018)

Auf Basis dieser Kriterien ergeben sich folgende nicht-funktionalen Anforderungen an die Process-Mining-Bibliothek:

- Die Bibliothek besitzt eine Verfügbarkeit von mindestens 99%.

- Die Bibliothek kann mit fehlerhaften Eventdaten arbeiten, ohne einen Fehlerzustand hervorzurufen.
- Die Bibliothek antwortet in 95% aller Fälle innerhalb von 10 Sekunden.
- Die Bibliothek muss in der Lage sein, mit Cases bestehend aus 100.000 Events arbeiten zu können.
- Die Bibliothek bietet eine grafische Oberfläche.
- Die Bibliothek kann innerhalb von zwei Tagen mit einer neuen Funktionalität erweitert werden.
- Die Bibliothek muss innerhalb einer Woche von anderen Entwicklern verstanden und gewartet werden.
- Die Bibliothek muss ohne vorherige Einarbeitung innerhalb von 10 Minuten zu benutzen sein.
- Die Bibliothek muss die verarbeiteten Eventdaten vor Unbefugten schützen.
- Die Bibliothek muss per REST-API angesprochen werden können.

3.2 Entwurf

Um aus den unsortierten Event-Daten im JSON-Format ein Petri-Netz erzeugen zu können, sind mehrere Schritte notwendig:

1. Die unsortierten Eventdaten müssen in einer geeigneten Datenstruktur abgelegt werden, sodass am Ende alle Events zu einem Case in chronologischer Reihenfolge vorhanden sind.
2. Die sortierte Datenstruktur wird verwendet, um ein Eventlog nach XES-Standard zu erzeugen.
3. Der α-Algorithmus wird verwendet, um auf Basis des generierten XES-Logs ein Petri-Netz zu erzeugen.
4. Das Petri-Netz als Datenstruktur muss zur Visualisierung in ein PNML-File umgewandelt werden.

Die Bibliothek besteht aus fünf Komponenten, vier davon setzen einen der oben genannten Schritte um. Zusätzlich gibt es eine Komponente, welche die HTTP-Requests verwaltet. Die Namen der Komponenten sind wie folgt:

1. Process-Mining-Controller
2. JSON-Reader
3. XES-Generator
4. Alpha-Miner
5. PNML-Generator

Das Zusammenspiel der einzelnen Komponenten kann in Abbildung 3.2 nachvollzogen werden. Die Bibliothek bietet eine REST-API an, über die mit einem POST-Request an /alphaMining, welcher die unsortierten Event-Daten im JSON-Format im Request-Body beinhaltet, die PNML-File zur Visualisierung eines Petri-Netzes angefragt werden kann. Die Anfrage wird zuerst an den Process-Mining-Controller delegiert, welcher nacheinander die benötigten Komponenten anspricht.

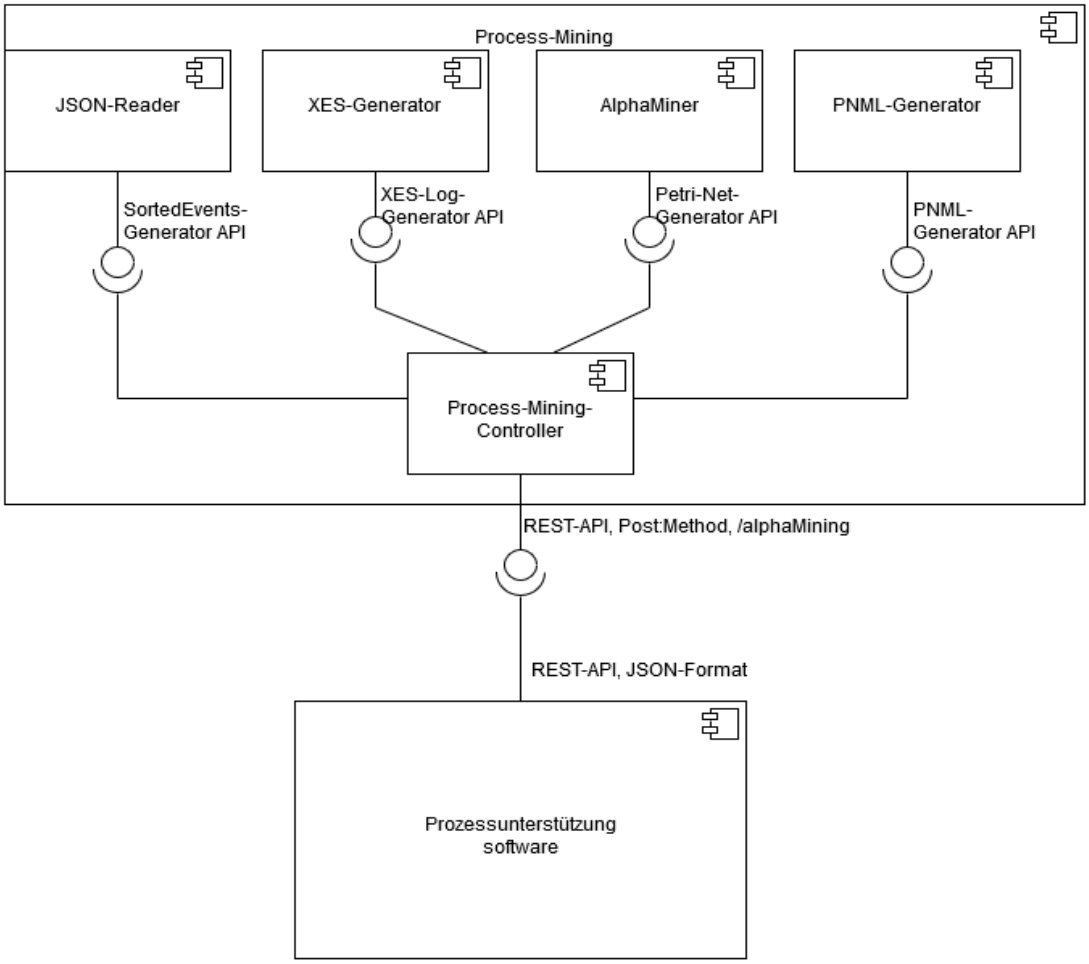

Abbildung 3.2: Komponenten-Diagramm

Um den Ablauf des POST-Request besser zu verstehen, kann das Sequenz-Diagramm in Abbildung 3.3 herangezogen werden. Der Anwender der Prozessunterstützungssoftware sendet einen POST-Request an /alphaMining, welcher bei dem Process-Mining-Controller eintrifft. Dieser delegiert nun die zu erledigenden Aufgaben an die jeweiligen Komponenten.

Zuerst werden die unsortierten Eventdaten an die Komponente JSON-Reader weitergereicht. Dort wird eine sortierte Datenstruktur erzeugt, in der alle Events zu einem Case in chronologischer Reihenfolge vorliegen, und an den Controller zurückgegeben. Diese Datenstruktur wird an die Komponente XES-Generator weitergegeben, welche ein Event-Log nach XES-Standard generiert. Anschließend erhält die Komponente AlphaMiner vom Controller das XES-Log und kann auf diesen den α-Algorithmus anwenden, um ein Petri-Netz zu erzeugen. Der Controller übergibt das erhaltene Petri-Netz an die Komponente PNML-Generator, sodass die PNML-File generiert werden kann. Anschließend wird die PNML-File an den Controller zurückgegeben und dieser leitet die PNML-File weiter an den Anwender.

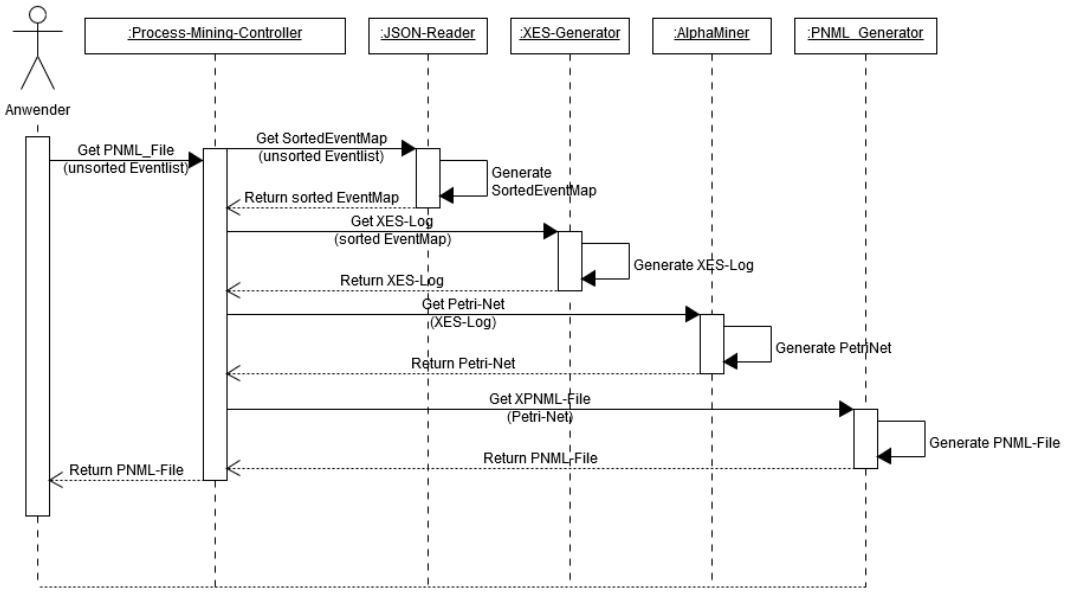

Abbildung 3.3: Sequenz-Diagramm

3.3 Implementation

Die Bibliothek wurde in der Programmiersprache Java entwickelt. Da die Bibliothek per REST-API angesprochen werden soll, wurde als Framework Spring Boot verwendet. Mit Hilfe von Spring Boot können schnell und einfach RESTful Web Services erstellt werden. Im Folgenden werden die Implementationen der einzelnen Komponenten genauer beschrieben.

3.3.1 JSON-Reader

Aufgabe dieser Komponente ist die Umwandlung der unsortierten Eventdaten im JSON-Format in eine geeignete Datenstruktur für die weitere Verarbeitung. JSON (JavaScript Object Notation) ist ein leichtgewichtiges Format, welches für den Austausch von Daten verwendet wird und im RFC 8259 standardisiert ist. JSON ist sowohl von Menschen als auch von Maschinen einfach zu lesen und zu schreiben. Darüber hinaus ist JSON programmiersprachenunabhängig. JSON basiert auf zwei Strukturen (vgl. Bray 2017):

- Eine Sammlung von Name- / Wert-Paaren. In verschiedenen Programmiersprachen wird dies als Objekt, Datensatz oder Struktur realisiert.
- Eine geordnete Liste von Werten. In den meisten Programmiersprachen wird dies als Array, Vektor, Liste oder Sequenz realisiert.

Damit der JSON-Reader mit den Eventdaten arbeiten kann, muss vorher eine Struktur festgelegt werden. In der Prozessunterstützungssoftware werden alle möglichen Events unsortiert gespeichert. Da also mehrere Events übergeben werden, wird eine Liste über Events benötigt. Ein Event besteht aus zwei Name-Wert-Paaren, die zwingend notwendig sind:

1. Name: CaseID, Wert: eindeutige Zahl, die den Case identifiziert. Jedes Event muss genau einem Case zugeordnet sein.
2. Name: EventAttributes, Wert: mehrere Name-Wert-Paare, welche Informationen über das Event beinhalten. Hier sind der Bezeichner und der Zeitpunkt des Events notwendig.

In Abbildung 3.4 ist ein Beispiel dieser Struktur zu sehen. Das Beispiel besteht aus drei Events A, B und C, welche alle der CaseID 1 zugeordnet sind. Zusätzlich zu den Eventnamen ist der Zeitpunkt und die Ressource (hier ein Sacharbeiter) abgespeichert. Die Bezeichnung dieser Attribute ist nach dem XES-Standard gewählt, welcher später genauer erläutert wird.

```
[
    {
        "caseID": "1",
        "eventAttributes":{
            "concept:name": "A",
            "time:timestamp": "100",
            "org:resource": "Alex"
        }
    },
    {
        "caseID": "1",
        "eventAttributes":{
            "concept:name": "B",
            "time:timestamp": "110",
            "org:resource": "Bert"
        }
    },
    {
        "caseID": "1",
        "eventAttributes":{
            "concept:name": "C",
            "time:timestamp": "120",
            "org:resource": "Cornelius"
        }
]
```

Abbildung 3.4: Beispiel JSON-Datei

Ursprünglich wurde angedacht eine externe Bibliothek zum Einlesen der JSON-Datei zu verwenden, aber da das Spring Boot Framework die Bibliothek Jackson integriert hat, wird keine externe Bibliothek mehr benötigt. Jackson mappt JSON-Daten auf eine POJO(Plain Old Java Object)-Klasse. Allgemein besteht eine POJO-Klasse nur aus Exemplarvariablen und den getter- und setter-Methoden. Die hierfür erstellte POJO-Klasse Event besteht aus einem String CaseID und aus einer Map<String,String> eventAttributes, welche die verschiedenen Informationen über ein Event speichern soll.

Schickt der Anwender nun eine Request mit JSON-Daten an /alphaMining, so mappt Jackson diese Eventdaten auf Objekte der POJO-Klasse Event. Da es sich um mehrere Events handelt, erhält man eine Liste<Event>. Diese Liste ist unsortiert und enthält die Events in der gleichen Reihenfolge wie in den JSON-Daten.

Anschließend muss aus dieser Liste die geeignete Datenstruktur erzeugt werden. Da die einzelnen Events einem Case zugeordnet werden müssen, bietet sich als Grundgerüst eine Map an, welche als Key die CaseID und als Wert eine Liste über Events beinhaltet. Da die CaseID als Key verwendet wird, besteht ein Event nur noch aus seinen Attributen, welche als Map<String,String> dargestellt werden können. Insgesamt erhält man als Datenstruktur eine Map<String, List<Map<String,String>>

Die Komponente JSON-Reader iteriert über diese Liste und sortiert die Events anhand der CaseID in die erstellte Datenstruktur ein. Alle einem Case zugehörigen Events sind nun in einer List vereint, aber liegen noch in unsortierter Reihenfolge vor. Mit Hilfe eines Comporators wird diese Liste mittels der Zeitpunkte der Events in chronologische Reihenfolge gebracht. Diese sortierte Datenstruktur kann nun von der Komponente XES-Generator verwendet werden.

3.3.2 XES-Generator

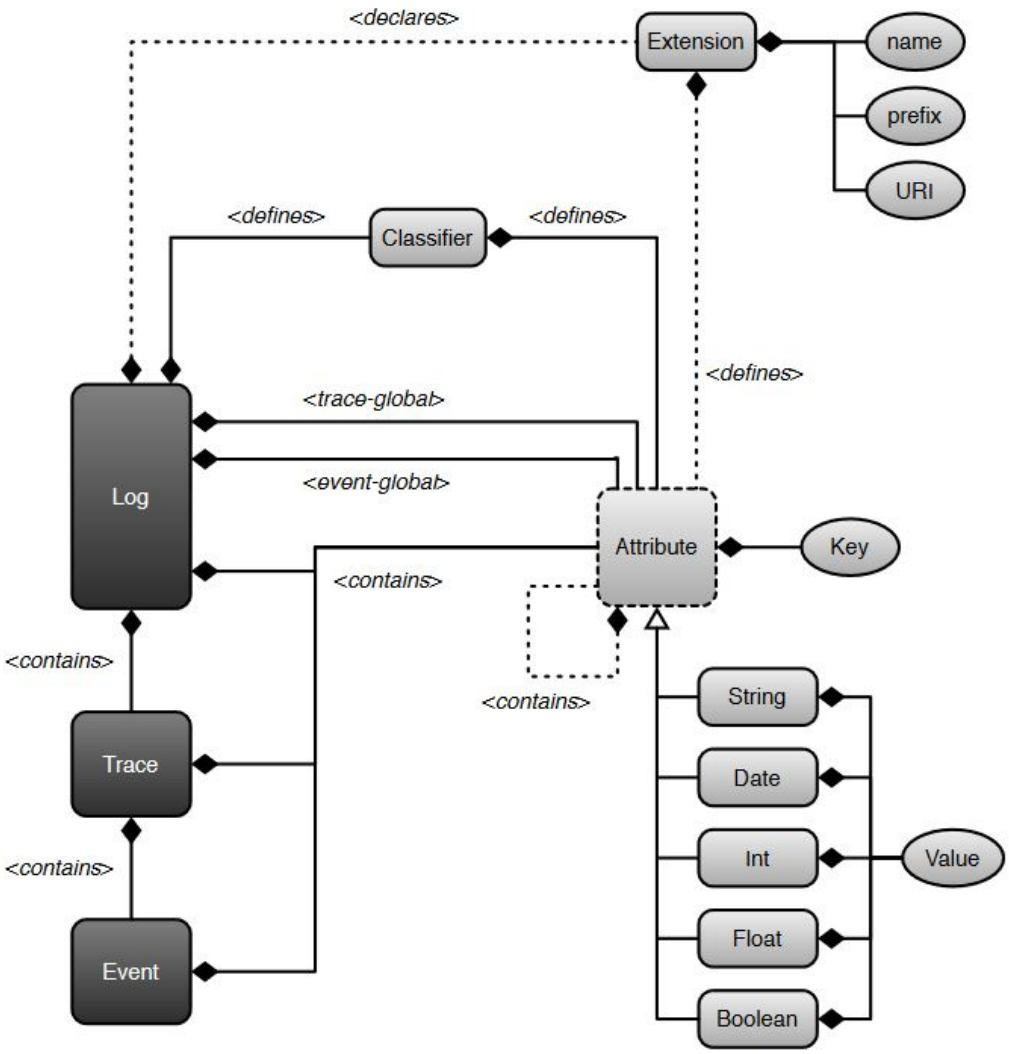

Abbildung 3.5: XES-Metamodell (Günther 2009, S.2)

Aufgabe dieser Komponente ist die Erzeugung eines Eventlogs nach XES-Standard aus der Datenstruktur, welche von dem JSON-Reader erstellt wurde.

Eventlogs können eine Vielzahl unterschiedlicher Formen annehmen. Jede Systemarchitektur, die einen Loggingmechanismus enthält, hat für diese Aufgabe eine eigene Lösung entwickelt. Um alle Arten von Ereignisprotokollen angemessen und mit möglichst gerin-

ger Verzerrung der Semantik zu erfassen, verwendet das XES-Model einen generischen Ansatz (vgl. Günther und Verbeek 2018, S.2).

Nur die Elemente, die in praktisch jedem Eventlog identifiziert werden können, werden explizit durch die Norm definiert. Alle weiteren Informationen werden durch optionale Attribute umgesetzt, welche durch sogenannte externe Extensions standardisiert werden können.

Die Basisstruktur des XES-Metamodells gibt an, dass ein Log aus einer nicht-leeren Ansammlung von Traces und dass ein Trace aus einer nicht-leeren Liste von Events besteht. Die Traces in einem Log können ungeordnet sein, wohingegen die Events in einem Trace geordnet sein müssen. Leere Logs und/oder Traces sind nicht erlaubt (vgl. Günther und Verbeek 2018, S.4). In Abbildung 3.6 ist ein Grundgerüst eines XES-Logs zu sehen.

```
<log xes.version="2.0" xes.features="" openxes.version="2.0" xmlns="http://www.xes-
    standard.org/">
  <trace>
    <event/>
    <event/>
    <event/>
    <event/>
    <event/>
    <event/>
  </trace>
  <trace>
    ...
  </trace>
  ...
</log>
```

Abbildung 3.6: Grundstruktur eines XES-Logs (Günther und Verbeek 2018, S.4)

Log, Traces, Events und Attribute selbst können mit Attributen versehen werden. Die Attribute können acht verschiedene Typen annehmen (vgl. Günther und Verbeek 2018, S.6-7):

- string
- boolean
- int
- float
- date
- id
- list
- container

Nach dem Hinzufügen von Attributen sieht das XES-Log beispielwiese wie folgt aus:

```
<log xes.version="1.3" xes.features="nested-attributes" openxes.version="1.8" xmlns="
    http://www.xes-standard.org/">
  <string key="Creator" value="Fluxicon Nitro"/>
  <trace>
    <string key="Name" value="Case1280"/>
    <string key="Creator" value="Fluxicon Nitro"/>
    <event>
      <string key="Resource" value="FL"/>
      <date key="Timestamp" value="2010-03-15T07:59:00.000+02:00"/>
      <string key="Operation" value="Handle Email"/>
      <string key="Agent Position" value="FL"/>
      <string key="Customer ID" value="Customer 1074"/>
      <string key="Product" value="iPhone"/>
      <string key="Service Type" value="Product Assistance"/>
      <string key="Agent" value="Susi"/>
    </event>
    <event>
      ...
    </event>
    ...
  </trace>
  <trace>
    ...
  </trace>
  ...
</log>
```

Abbildung 3.7: XES-Logs mit Attributen (Günther und Verbeek 2018, S.8-9)

Da die Namen der Attribute bisher willkürlich gewählt werden konnten, ist noch keine Vergleichbarkeit zwischen verschiedenen Eventlogs gegeben. Damit aus diesem Modell ein Standard werden kann, bedarf es der sogenannten Extensions. Eine Extension bietet den beteiligten Attributen eine festgelegte Semantik. Durch die Deklaration einer Extension

im Log und die Verwendung der von dieser Extension vorgegebenen Attributschlüssel, gibt der Ersteller des Logs an, dass die Semantik des jeweiligen Attributwertes, der von der Erweiterung vorgegebenen Semantik entspricht (vgl. Günther und Verbeek 2018, S.13). Die Extensions funktionieren daher wie ein Vertrag, den die Verwender eingehen, und haben das Ziel eine standardisierte Namensgebung der Attributschlüssel zu schaffen.

Im Folgenden werden die wichtigsten Extensions und ihre Verwendungsgebiete benannt (vgl. Günther und Verbeek 2018, S.15-18):

- Concept: Wird genutzt, um den Elementen einen Namen zu geben. Auf Log-Ebene kann z.B. der Prozessname gespeichert werden. Bei Traces wird normalerweise die CaseID gespeichert. Für Events wird der Name der ausgeführten Aktivität gespeichert.
- Lifecycle: Hier können einzelnen Events Zustände wie z.B. gestartet, pausiert, wieder aufgenommen oder beendet zugeordnet werden.
- Organizational: Jedem Event kann eine ausführende Ressource, Rolle oder Gruppe zugewiesen werden.
- Time: Wird genutzt, um anzugeben, an welchem Datum und/oder Uhrzeit ein Event stattgefunden hat.
- Cost: Einzelnen Events oder kompletten Traces können Kosten zugeordnet werden.

Das XES-Log mit Extensions sieht folgendermaßen aus:

```
<log xes.version="1.3" xes.features="nested-attributes" openxes.version="1.6" xmlns="
    http://www.xes-standard.org/">
  <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.
      xesext"/>
  <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
  <extension name="Organizational" prefix="org" uri="http://www.xes-standard.org/org.
      xesext"/>
  <global scope="trace">
    <string key="concept:name" value="name"/>
  </global>
  <global scope="event">
    <string key="concept:name" value="Handle Email-Product Assistance"/>
    <string key="org:resource" value="resource"/>
    <date key="time:timestamp" value="2010-10-07T21:56:18.312+02:00"/>
    <string key="Operation" value="string"/>
    ...
    <string key="Service Type" value="string"/>
  </global>
```

Abbildung 3.8: XES-Logs mit Attributen (Günther und Verbeek 2018, S.14)

Neben den oben geschilderten Möglichkeiten bietet der XES-Standard zusätzlich globale Attribute und Classifier an.

Die globalen Attribute können genutzt werden, um anzugeben, dass bestimmte Attribute verpflichtend sind. Beispielhaft könnte man angeben, dass jeder Trace einen Namen (TraceID) oder dass jedes Event einen Namen, Zeitstempel und ausführende Ressource haben muss (vgl. Aalst 2016, S. 139,141).

Die Classifier stellen eine Aggregationsmöglichkeit für Events dar. Ein Classifier beinhaltet eine Liste von Attributschlüsseln, auf dessen Basis eine Zusammenfassung des Logs durchgeführt werden kann. Diese Zusammenfassung kann Informationen darüber beinhalten, wie viele verschiedene Kombinationen des jeweiligen Classifiers im Log enthalten sind und wie häufig zu auftreten. Ein Beispiel für einen Classifier könnte z.B. die Kombination Eventname und ausführende Ressource sein (vgl. Günther und Verbeek 2018, S.9).

Nachdem der XES-Standard genauer vorgestellt wurde, wird nun die direkte Implementation beschrieben. Die vom JSON-Reader erzeugte Datenstruktur muss in ein XES-Log umgewandelt werden. Für diese Umwandlung wird die OpenXES-Bibliothek verwendet, welche einfach zu nutzen und konform mit dem XES-Standard in jedem Aspekt ist.

Die von der OpenXES-Bibliothek erzeugten Elemente sind von der Namensgebung hier fast komplett identisch mit dem XES-Standard. Der einzige Unterschied besteht darin,

dass alle Elemente den Prefix "X" haben. Ein Log heißt beispielweise XLog.

Um das XES-Log zu erstellen sind nun folgende Schritte notwendig:

1. Erstelle ein XLog.
2. Füge dem XLog die notwendigen Extensions hinzu. In diesem Fall sind es die Extensions Concept, Time und Organizational.
3. Iteriere über die Traces in der Datenstruktur. Erzeuge jedes Mal ein XTrace und füge es dem XLog hinzu.
4. Iteriere für jeden Trace über alle in ihm enthaltenen Events. Erzeuge jedes Mal ein XEvent und füge es dem jeweiligen XTrace hinzu.
5. Iteriere für jedes Event über alle in ihm enthaltenen Attribute. Erzeuge jedes Mal ein XAttribute und füge es dem jeweiligen Event hinzu.
6. Verwende den XesXML-Serializer um das Log zu serialisieren.

Ein Ausschnitt eines Eventlogs, welches von der OpenXES-Bibliothek erzeugt worden ist, ist in Abbildung 3.9 zu sehen:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This file has been generated with the OpenXES library. It conforms -->
<!-- to the XML serialization of the XES standard for log storage and -->
<!-- management. -->
<!-- XES standard version: 1.0 -->
<!-- OpenXES library version: 1.0RC7 -->
<!-- OpenXES is available from http://www.openxes.org/ -->
<log xes.version="1.0" xes.features="nested-attributes" openxes.version="1.0RC7">
    <extension name="Organizational" prefix="org" uri="http://www.xes-standard.org/org.xesext"/>
    <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
    <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
    <global scope="trace">
        <string key="concept:name" value="UNKNOWN"/>
    </global>
    <global scope="event">
        <string key="concept:name" value="UNKNOWN"/>
        <string key="time:timestamp" value="UNKNOWN"/>
        <string key="org:resource" value="UNKNOWN"/>
    </global>
    <trace>
        <string key="concept:name" value="1"/>
        <event>
            <string key="concept:name" value="A"/>
            <string key="time:timestamp" value="100"/>
            <string key="org:resource" value="Alex"/>
        </event>
        <event>
            <string key="concept:name" value="B"/>
            <string key="time:timestamp" value="110"/>
            <string key="org:resource" value="Alex"/>
        </event>
        <event>
            <string key="concept:name" value="C"/>
            <string key="time:timestamp" value="120"/>
            <string key="org:resource" value="Alex"/>
        </event>
        <event>
            <string key="concept:name" value="D"/>
            <string key="time:timestamp" value="130"/>
            <string key="org:resource" value="Alex"/>
        </event>
    </trace>
```

Abbildung 3.9: Eventlog erzeugt mit OpenXES-Bibliothek

3.3.3 AlphaMiner

Die Aufgabe der Komponente AlphaMiner ist die Erzeugung eines Petri-Netzes mit Hilfe des XES-Logs. Die Funktionsweise des verwendeten α-Algorithmus wurde in Abschnitt 2.3 bereits genauer erläutert. In diesem Abschnitt wird das UML-Klassendiagramm dieser Komponente vorgestellt und die notwendigen Schritte zur Erzeugung eines Petri-Netzes beschrieben.

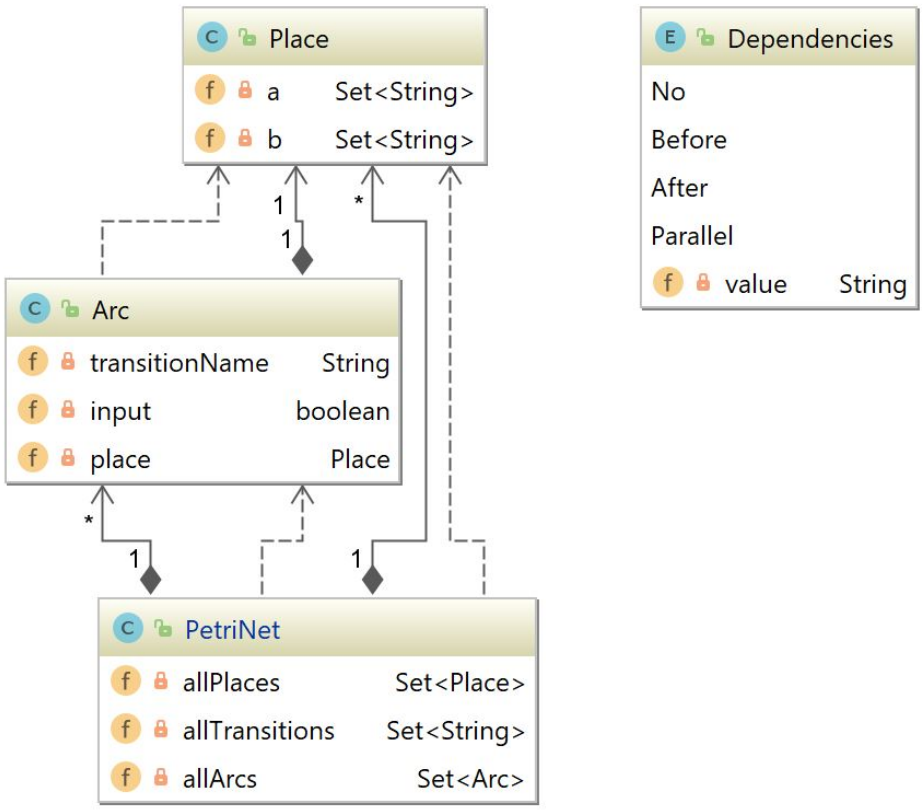

Abbildung 3.10: UML-Klassendiagramm AlphaMiner

UML-Klassendiagramm

Abbildung 3.10 zeigt das UML-Klassendiagramm der Komponente AlphaMiner. Ein Petri-Netz besteht insgesamt aus drei Teilen: Places, Transitions und Arcs. Da eine Transition nur eine Beschreibung einer Aktivität ist, wird diese als String dargestellt. Das führt dazu, dass die Klasse PetriNet aus einem Set über Places, einem Set über Strings (stellvertretend für die Transitions) und einem Set über Arcs besteht.

Ein PetriNet kann aus einem oder mehreren Places bestehen. Ein Place setzt sich wiederum aus zwei Sets über String (stellvertretend für die Transitions) zusammen.
Ein PetriNet kann aus einem oder mehreren Arcs bestehen. Ein Arc besteht aus einem String für die Transition und einem Place. Außerdem besitzt ein Arc einen boolean, der angibt, ob es sich um einen Input- oder Output-Arc handelt.

Des Weiteren beinhaltet die Komponente das Enum Dependencies, welches für die Erzeugung der Footprint-Matrix genutzt wird.

Darüber hinaus gibt es die Klasse AlphaAlgorithm, die auf alle anderen Klassen zugreift und alle notwendigen Schritte zu Erzeugung des Petri-Netzes ausführt:

1. Überprüfe, welche Events im Eventlog vorhanden sind. Diese Events werden die Transitions des Petri-Netzes.
2. Ermittle alle Events, die am Anfang eines Traces stattgefunden haben.
3. Ermittle alle Events, die am Ende eines Traces stattgefunden haben.
4. Erstelle die Footprint-Matrix.
5. Erzeuge alle non-maximal Places.
6. Erzeuge alle maximal Places.
7. Erzeuge die Start- und Endplaces.
8. Erzeuge alle Arcs.
9. Erstelle aus den Transitions, Places und Arcs das Petri-Netz.

Mit Hilfe des erzeugten Petri-Netzes kann anschließend eine sogenannte PNML-File erzeugt werden.

3.3.4 PNML-Generator

Die Aufgabe der Komponente PNML-Generator ist die Generierung einer PNML-File aus dem erzeugten Petri-Netz. Diese PNML-File kann von Bibliotheken oder Tools, wie z.B. ProM verwendet werden, um den zugrundeliegenden Prozess zu visualisieren. Im folgenden Abschnitt wird der Aufbau einer PNML-File erläutert und die Verwendung einer Bibliothek zur Erzeugung einer solchen PNML-File beschrieben.

Die Abkürzung PNML steht für PetriNet-Markup-Language, welches ein XML-basiertes Austauschformat für PetriNet-Tools ist, damit diese PetriNet-Modelle austauschen können. PNML ist zurzeit standardisiert in ISO/IEC-15909 (vgl. Hillah und Ekkart 2009, S.5)

Grundsätzlich kann jede Art von Petri-Netz durch die folgenden Bestandteile beschrieben werden (vgl. Hillah und Ekkart 2009, S.17):

- Transitionen
- Places
- Arcs

Zusätzlich können diese Bestandteile mit Labels versehen werden, um die Verständlichkeit des Modells zu erhöhen.

In Abbildung 3.11 ist ein Beispiel einer PNML-File zu sehen. Die Reihenfolge der einzelnen Elemente ist dabei unwesentlich und muss nicht den logischen Ablauf eines Prozesses widerspiegeln; die Elemente könnten auch sequentiell abgearbeitet werden, d.h., erst die Transitionen, anschließend die Places und zuletzt die Arcs.

```
<pnml xmlns="http:...">
 <net id="n1" type="...">
  <page id="pg1">
   <place id="p1"/>
   <arc id="a1" source="p1"
                target="t1"/>
   <transition id="t1"/>
   <arc id="a2" source="t1"
                target="p2"/>
   <place id="p2"/>
  </page>
 </net>
</pnml>
```

Abbildung 3.11: Beispiel PNML-File (Hillah und Ekkart 2009, S.36)

Um eine solche PNML-File zu erzeugen, kann das PNML-Framework verwendet werden. Alle vom PNML-Framework erzeugten Objekte enden auf dem Suffix "HLAPI". Folgende Schritte sind notwendig zur Erzeugung einer PNML-File:

1. Erzeuge ein PetriNetDocumentHLAPI. Dies ist der äußerste Rahmen der PNML-File.

2. Erzeuge ein PetriNetHLAPI. Das PetriNet wird einem PetriNetDocumentHLAPI zugewiesen; es können mehrere PetriNets einem PetriNetDocumentHLAPI zugewiesen werden.

3. Erzeuge eine PageHLAPI. Die PageHLAPI wird einem PetriNetHLAPI zugewiesen. Die Page ist notwendig, da sie als Container für die Transitionen, Places und Arcs dient.

4. Iteriere über alle Transitionen des vom AlphaMiner generierten PetriNets und erzeuge eine TransitionHLAPI. Füge die TransitionHLAPI der PageHLAPI hinzu und speichere sie zwischen, da sie für die Arcs benötigt wird.

5. Iteriere über alle Places des vom AlphaMiner generierten PetriNets und erzeuge eine PlaceHLAPI. Füge die PlaceHLAPI der PageHLAPI hinzu und speichere sie zwischen, da sie für die Arcs benötigt wird.

6. Erzeuge mit Hilfe der TransitionHLAPI und PlaceHLAPI die ArcHLAPI und füge sie der PageHLAPI hinzu. Bei der Erzeugung muss darauf geachtet werden, ob der Arc ein Input- oder Output-Arc ist. Handelt es sich um einen Input-Arc, muss als erster Parameter die TransitionHLAPI und als zweiter Parameter die PlaceHLAPI übergeben werden, bei einem Output-Arc muss die Reihenfolge vertauscht werden.

7. Exportiere das PetriNetDocumentHLAPI und erzeuge somit die PNML-File.

3.4 Testen und Resultate

3.4.1 Testen

Zuerst wurde ein Komponententest durchgeführt. Dabei wurde überprüft, ob die einzelnen Komponenten das erwartete Ergebnis zurückliefern:

- JSON-Reader: Datenstruktur, in der alle zu einem Case gehörigen Events in chronologischer Reihenfolge sortiert sind.
- XES-Generator: Eventlog, welches dem XES-Standard entspricht.
- AlphaMiner: Petri-Netz als Datenstruktur, bestehend aus Transitionen, Places und Arcs.
- PNML-Generator: File, welche dem PNML-Standard entspricht.

Nachdem die Funktionsweise der einzelnen Komponenten separat überprüft worden sind, wurde ein Integrationstest durchgeführt. Dabei wurde der Output einer Komponente als Input an die jeweilige nachfolgende Komponente übergeben.

Da die Bibliothek im Produktionsbetrieb per REST-API angesprochen werden soll, wurde ein Systemtest durchgeführt. Zuerst wurde die Anwendung auf dem Port localhost:8080 gestartet. Für das Erstellen des REST-Request wurde das Tool *Postman* verwendet. Postman bietet eine grafische Oberfläche mit deren Hilfe einfach HTTP-Requests erstellt, verwaltet und verwendet werden können.

Anschließend wurde ein Request mit der Methode *Post* an die URL *http:localhost:8080/alphaMining* gesendet. Der Body der Request beinhaltete ein Eventlog im JSON-Format. Der Request kann in Abbildung 3.12 betrachtet werden.

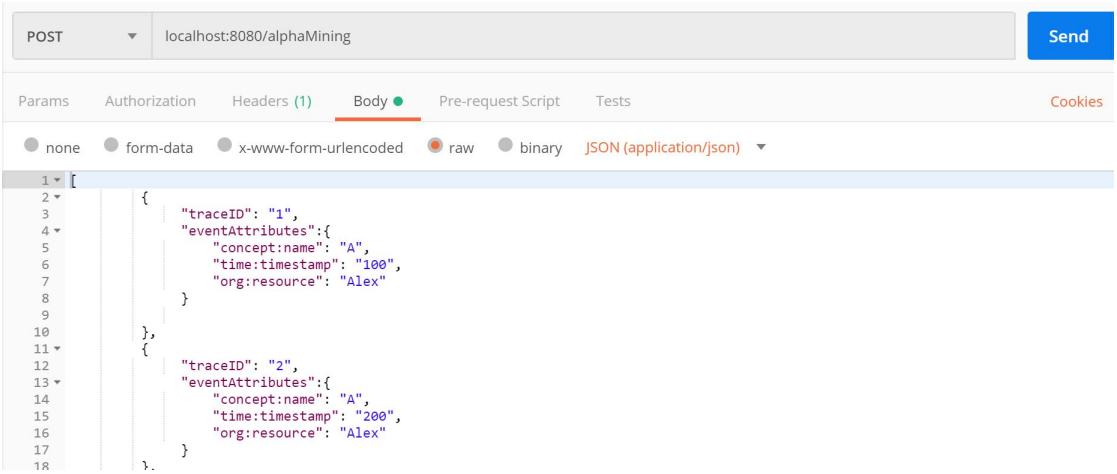

Abbildung 3.12: Postman Request

Der Response beinhaltet in seinem Body die PNML-File und sieht für das in diesem Beispiel verwendete Eventlog L2 folgendermaßen aus:

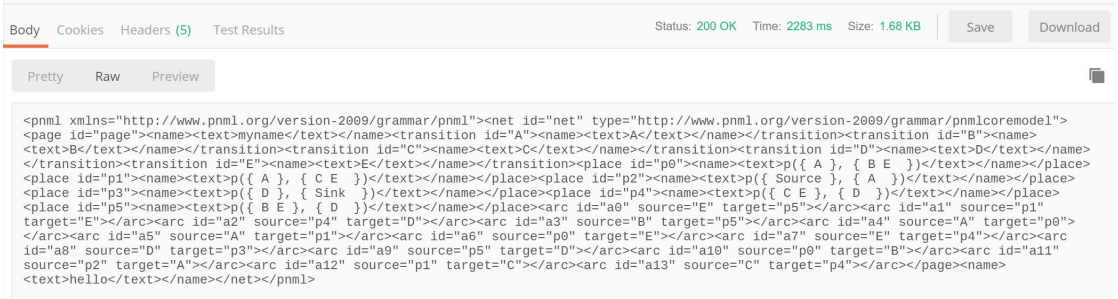

Abbildung 3.13: Postman Response

3.4.2 Resultate

Das Überprüfen der Resultate hat sich als schwierig erwiesen, da kein vorhersagbares Ergebnis existiert. Process-Mining wird angewendet, um z.B. aus einem Eventlog ein Prozess-Modell zur Visualisierung zu gewinnen und Einblicke in den zugrundeliegenden Prozess zu erhalten; es hat einen explorativen Charakter. Würde das Ergebnis im Vorfeld bekannt sein, wäre die Anwendung von Process-Mining in diesem Kontext überflüssig. Folglich ließen sich keine klassischen *expected,got*-Tests durchführen.

Ein weiteres Problem bei dem Überprüfen der Resultate war, dass das gewünschte Endergebnis eine Visualisierung ist. Visualisierungen können nur schwer von Maschinen auf Richtigkeit getestet werden und darüber hinaus können mehrere richtige Visualisierungen für einen Prozess existieren. Um die Funktionsweise der Anwendung trotzdem zu prüfen, würden die Ergebnisse der Anwendung mit denen aus dem Buch *Process Mining - Data Science in Action* verglichen (vgl. Aalst 2016, S.163-174). Dabei wurde einerseits die formale Beschreibung der Petri-Netze und andererseits die erzeugten Petri-Netz-Modelle auf Übereinstimmung geprüft.

Um den Vergleich durchzuführen, wurden folgende vier Eventlogs aus dem Buch rekonstruiert:

L_1 = [(a, b, c, d)^3, (a, c, b, d)^2, (a, e, d)]

L_2 = [(a, b, c, d)^3, (a, c, b, d)^4, (a, b, c, e, f, b, c, d)^2, (a, b, c, e, f, c, b, d)
(a, c, b, e, f, b, c, d)^2, (a, c, b, e, f, b, c, e, f, c, b, d)]

L_3 = [(a, b, c, d, e, f, b, d, c, e, g), (a, b, d, c, e, g)^2,
(a, b, c, d, e, f, b, c, d, e, f, b, d, c, e, g)]

L_4 = [(a, c, d)^{45}, (b, c, d)^{42} (a, c, e)^{38}, (b, c, e)^{22}]

Die formalen Beschreibungen stimmten bei allen vier Logs überein, bei den Visualisierungen gab es einzig bei dem Eventlog L_2 eine Abweichung.

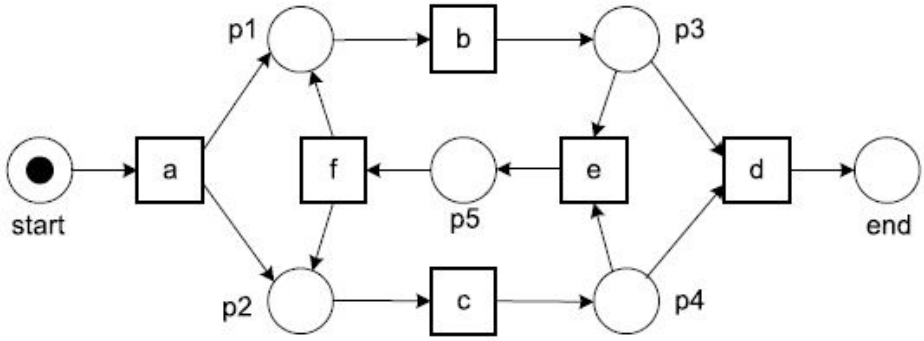

Abbildung 3.14: Visualisierung des Eventlogs L_2 aus Process Mining - Datascience in Action (Aalst 2016, S.165)

Wie bei dieser Visualisierung zu erkennen ist, handelt es sich um einen komplexeren

Prozess, welcher eine Schleifen-Konstruktion beinhaltet. Die Visualisierung der von der Anwendung erzeugten PNML-File sieht im Vergleich folgendermaßen aus:

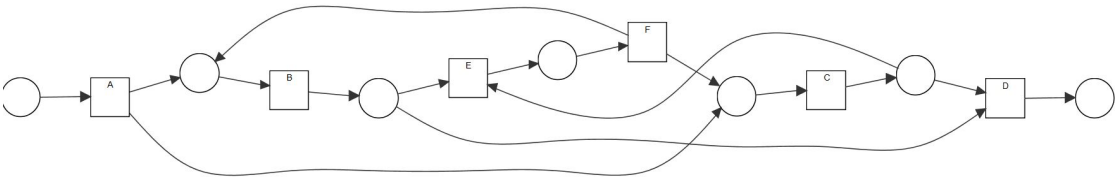

Abbildung 3.15: Visualisierung des Eventlogs L_2 auf Basis einer von der Anwendung erzeugten PNML-File

Auf dem ersten Blick sieht die Visualisierung in Abbildung 3.15 komplett unterschiedlich aus, aber tatsächlich beschreiben beide Varianten ein und denselben Prozess. Die Ursache für diese Abweichung ist, dass die Anwendung den Elementen in der PNML-File keine genaue Position in Form von X- und Y-Koordinaten übergibt. Diese müssten zuerst mit Hilfe von Graph-Visualisierungs-Algorithmen berechnet werden. Die Aufgabe der Positionierung der Elemente wird daher vollständig den Visualisierungs-Tools überlassen und führt bei komplexeren Prozess-Modellen zu Abweichungen.

3.5 Deployment

Die Bibliothek wurde bisher nur lokal gestartet und getestet. Damit die Bibliothek von anderen Anwendungen, wie z.B. der Prozess-Unterstützungssoftware, benutzt werden kann, muss sie öffentlich zugänglich gemacht werden. Für das Deployment wurde die ICC (Informatic-Compute-Cloud) der HAW genutzt. Die ICC stellt eine Umgebung zur Ausführung von Docker-Containern in einem Kubernetes-Cluster zur Verfügung.

Um von einer lokalen Bibliothek zu einem lauffähigen Docker-Container in der ICC zu kommen, muss zuerst ein Dockerfile erstellt werden. Ein Dockerfile kann als eine Blaupause verstanden werden, nach der ein Docker-Image erstellt wird. Aus diesem Docker-Image können anschließend konkrete Instanzen, die Docker-Container, erzeugt werden. Die Dockerfile besteht aus folgenden Bestandteilen:

- FROM openjdk:13-jdk-alpine. Dies ist das Basis-Image für den Docker-Container.

- VOLUME /tmp. Volumes sind der bevorzugte Mechanismus zum Speichern von Daten, die von Docker-Containern generiert und von diesen verwendet werden. Der Befehl erzeugt ein Volume namens tmp im Docker-Container.
- EXPOSE 8080. Der in Spring-Boot integrierte Tomcat-Server startet standardmäßig auf dem Port 8080. Damit die Anwendung verwendet werden kann, muss dieser Port außerhalb des Containers zugänglich gemacht werden.
- Copy . /usr/src/app . Copiert das komplette Projektverzeichnis in den Ordner /usr/src/app des Containers.
- CMD ["java", -"jar", "/usr/src/app/processmining.jar"]. Bei Start des Containers wird die JAR-Datei processmining.jar ausgeführt.

Die im CMD-Befehl verwendete JAR-Datei muss zuvor generiert werden. Diese wird mit Hilfe des Build-Management-Tools *Apache Maven* erstellt.

Mit Hilfe der Dockerfile kann ein Docker-Container gestartet werden. Um diesen Docker-Container in dem Kubernetes-Cluster der ICC zu starten, bedarf es einer YAML-Datei namens deploy.yaml (vgl. Stäps und Blanck 2019, S. 13). Damit bei einer Änderung der Bibliothek das Deployment nicht jedes Mal manuell durchgeführt werden muss, kann mit Hilfe einer .gitlab-ci.yml Datei das Deployment automatisiert werden (vgl. Stäps und Blanck 2019, S. 17). Sobald eine Änderung des Projektes committed wird, wird das Docker-Image neu gebaut und anschließend ein Docker-Container in der ICC gestartet.

Ob die Bibliothek fehlerfrei in der ICC läuft, kann mit Hilfe von Port-Forwarding getestet werden. Anfragen an localhost:8080 werden an den Port 8080 des im ICC gestarteten Docker-Container weitergeleitet. Die Funktionalität der Bibliothek kann anschließend, wie im Abschnitt 3.4 beschrieben, getestet werden.

3.6 Erweiterbarkeit

In diesem Abschnitt werden der technische und fachliche Aspekt der Erweiterbarkeit beschrieben.

3.6.1 Technische Aspekte

Zurzeit bietet die Bibliothek nur die Möglichkeit den α-Algorithmus auf ein Eventlog anzuwenden, um eine PNML-File zur Visualisierung des zugrundliegenden Prozesses zu erhalten. Um weitere Funktionalitäten bedienen zu können, müssen in dem Process-Mining-Controller neue Routes hinzugefügt werden. Die Idee dahinter besteht darin, dass viele kleine Handler für diese Routes existieren, die auf die Komponenten der Bibliothek zugreifen und die Outputs der Komponenten solange als Input an andere Komponenten weiterreichen, bis das vom Anwender gewünschte Endergebnis erreicht worden ist.

Dabei ist zu beachten, dass die einzelnen Komponenten weiterhin eine hohe Kohäsion und eine geringe Kopplung vorweisen. Die Komponenten sollten in sich abgeschlossene Aufgaben übernehmen, deren Umfang überschaubar ist. Darüber hinaus kommunizieren die Komponenten nur über den Prozess-Mining-Controller, welcher als Aufgabenverteiler fungiert, und haben deshalb eine sehr geringe Kopplung.

Dies hat den Vorteil, dass einzelne Komponenten einfach ausgetauscht werden können. Sollte in Zukunft ein anderer Algorithmus anstelle des α-Algorithmus verwenden werden, so kann die alte Komponente problemlos ausgetauscht werden, sofern die neue Komponente die gleichen Schnittstellen anbietet. In diesem Fall müsste die neue Komponente auch ein XES-Log als Input entgegennehmen und ein Petri-Netz als Datenstruktur zurückgeben.

Außerdem bietet dieser modulare Aufbau die Möglichkeit bisherige Zwischenergebnisse an den Anwender zurückzugeben. Ist der Anwender beispielsweise nur an einem XES-Log interessiert, könnte die neue Route /XES-Log hinzugefügt und in dessen Handler nur die Komponenten JSON-Reader und XES-Log verwendet werden.

3.6.2 Fachliche Aspekte

Der bisher angebotene α-Algorithmus bringt gewisse Einschränkungen mit sich, wie z.B. der Umgang mit Schleifen einer Länge kleiner als drei. Um ein Prozess-Modell ohne Einschränkungen entdecken zu können, müssen fortgeschrittenere Algorithmen, wie die Erweiterung des α-Algorithmus, der α^+-Algorithmus, Genetic Mining oder Inductive Mining verwendet werden. Diese Algorithmen könnten dann z.B. über die Routes /alphaMining+, /geneticMining und /inductiveMining angesprochen werden.

Des Weiteren könnte man Komponenten hinzufügen, die Petri-Netze in andere Modellierungssprachen, wie z.B. BPNM oder EPK, umwandeln.

Die bisher genannten Komponenten sind alle nur Process-Discovery zuzuordnen, welches nur eines von drei Teilgebieten von Process-Mining ist. Damit aus dieser Bibliothek eine vollständige Process-Mining-Bibliothek wird, müssen die anderen beiden Teilgebiete, Process-Conformance und Process-Enhancement, abgedeckt werden.

Bei Process-Conformance könnte man Komponenten anbieten, die die Fitness von einem Eventlog und Prozess-Modell berechnen. Dabei könnten verschiedene Methoden wie Token-Replay oder Alignment verwendet werden.

Bezüglich Process-Enhancement existieren die Möglichkeiten Repair und Extension. Repair versucht ein bestehendes Prozess-Modell mehr an die Realität anzupassen. Unter Extension versteht man das Ergänzen des Prozess-Modells um weitere Perspektiven, indem man von zusätzlichen Attributen im Eventlog, wie z.B. der Ressource oder dem Zeitstempel, Gebrauch macht. Mit diesen Perspektiven könnten die Beziehungen zwischen den einzelnen Ressourcen modelliert oder Engpässe bzw. kritische Pfade untersucht werden (vgl. Aalst 2016, S.33)

Wird die Bibliothek mit den Teilgebieten Process-Conformance und Process-Enhancement erweitert, sollte das Routing der Bibliothek überarbeitet werden. Damit die Bibliothek nicht an Übersichtlichkeit verliert, bietet es sich an, die Funktionen den jeweiligen Teilgebieten zuzuordnen, sodass sich beispielweise die Route /alphaMining zu /processDiscovery/alphaMining verändert.

4 Fazit

Prozesse sind ein wesentlicher Bestandteil der heutigen Welt, die Dienstleistungen und interne Funktionen in Unternehmen, Behörden und Organisationen auf der ganzen Welt vorantreibt. Zwar gibt es viele Systeme, die die Ausführung solcher Prozesse unterstützen, jedoch lassen die derzeitigen Praktiken zur Überwachung und Analyse dieser Ausführung in der Realität immer noch zu wünschen übrig. Process-Mining ist in der Lage, diese Lücke zu schließen und bietet Mittel für die Analyse und Überwachung realer Prozesse.

Der Ablauf von Process-Mining kann in fünf Schritte unterteilt werden:

1. Erhalte ein Eventlog: In Abschnitt 2.1 wurde beschrieben, aus welchen Quellen man Daten extrahieren kann und wie ein Eventlog aufgebaut sein muss.
2. Kreiere oder entdecke ein Prozess-Modell: In Abschnitt 2.2 und 2.3 wurden Techniken des Process-Discovery vorgestellt, wie beispielsweise Alpha-Mining, Heuristic-Mining oder Genetic-Mining.
3. Verbinde die Events im Eventlog mit den Aktivitäten im Prozess-Modell. Die Token-Replay-Technik, beschrieben in Abschnitt 2.4, stellt beispielsweise diese Verbindung her.
4. Erweitere das Prozess-Modell. In Abschnitt 2.5 würde erläutert, wie dem Prozess-Modell weitere Perspektiven (Organisation-, Zeit- und Caseperspektive) hinzugefügt werden können.
5. Gebe das integrierte Prozess-Modell zurück.

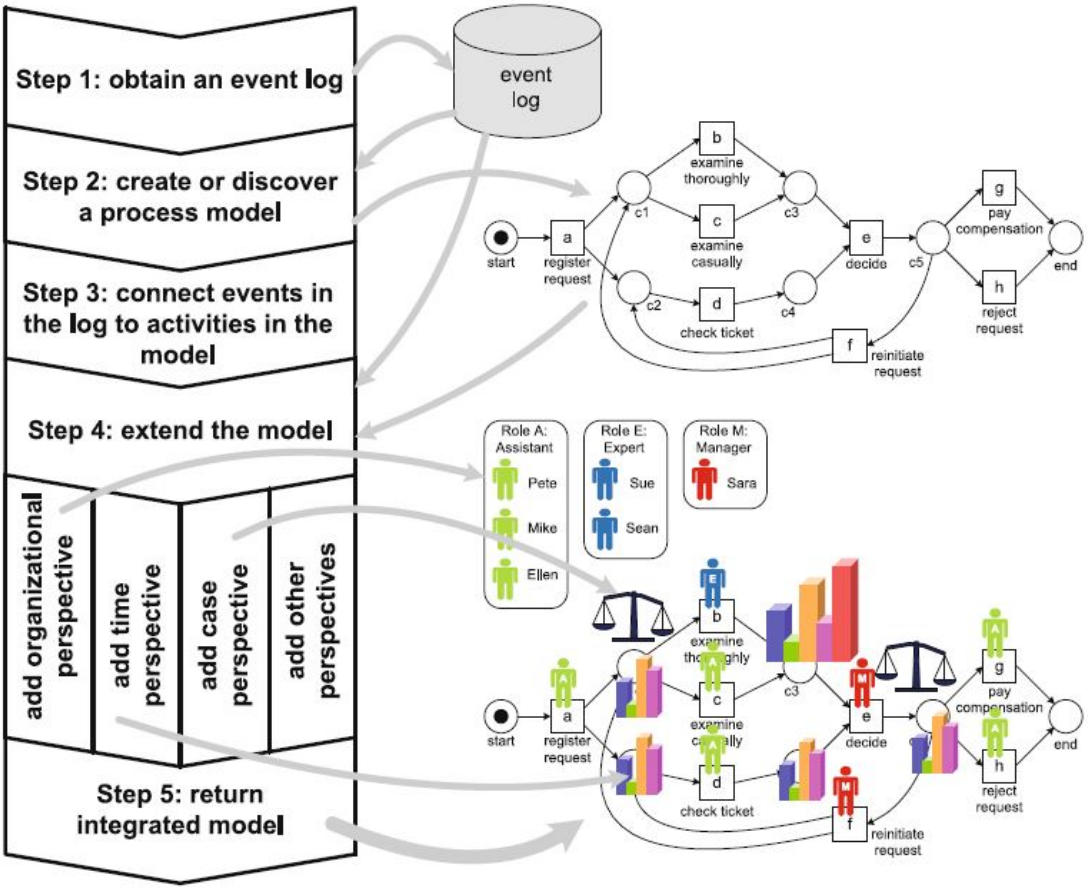

Abbildung 4.1: Ablauf Process-Mining (Aalst 2016, S.298)

In Rahmen dieser Bachelorarbeit wurden Process-Mining-Techniken vorgestellt, welche vergangenheitsorientiert arbeiten, d.h. auf abgeschlossenen Cases. Diese Techniken können in die drei Teilbereiche Process-Discovery, Process-Conformance und Process-Checking unterteilt werden.

Im praktischen Teil dieser Arbeit wurde der Grundstein für eine Bibliothek zur generischen Anwendung von Process-Mining gelegt. Auf dem Markt existieren etliche Process-Mining-Anwendungen, diese müssen aber zuerst installiert werden und können vergleichsweise schwer in bestehende Anwendungen integriert werden. Der Mehrwert dieser Bibliothek besteht darin, dass man über eine REST-API leichten Zugang zu den Ergebnissen von Process-Mining erhält.

Bei der Entwicklung der Bibliothek wurde sehr darauf geachtet, dass die Inputs und

Outputs standardisiert sind (XES, PNML), damit eventuell notwendige Anpassungsarbeiten auf ein Minimum reduziert werden können. Außerdem wurde die Architektur so ausgelegt, dass in Zukunft weitere Komponenten ohne Probleme hinzugefügt werden.

Die Bibliothek umfasst zurzeit 5 Komponenten. Im Vergleich dazu verfügt das Process-Mining-Tool ProM über 1500 Plug-Ins. Dies zeigt, dass die Bibliothek nur einen winzigen Bruchteil des komplexen Themas Process-Mining und nicht alle in Abschnitt 3.1.1 gestellten funktionalen Anforderungen abdecken kann. Im Rahmen weiterer Abschlussarbeiten, könnte diese Bibliothek um weitere Funktionalitäten, wie in Abschnitt 3.6.2 beschrieben, erweitert werden.

Außerdem konnten auch nicht alle nicht-funktionalen Anforderungen, beschrieben in Abschnitt 3.1.2, umgesetzt werden. Zurzeit bietet die Bibliothek keine grafische Oberfläche und setzt voraus, dass die Benutzer wissen, welche Daten, mit welcher HTTP-Methode, an welchen Endpunkt gesendet werden müssen.

Es wäre wünschenswert, dass sich unter dem Start-Endpunkt eine Übersicht befindet, welche alle weiteren Endpunkte, gegliedert nach den drei Teilbereichen Process-Discovery, Process-Conformance und Process-Checking, auflistet. Wird einer dieser Endpunkte im Browser aufgerufen, sollte eine Seite erscheinen, die die Verwendung dieses Endpunktes genauer erläutert. Dazu gehört eine kurze Beschreibung, der notwendige Input, der zu erwartende Output und ein Beispiel, welches die Verwendung dieses Endpunktes zeigt.

Darüber hinaus kann das in Abschnitt 3.3.4 beschriebene PNML-Framework nicht mit Umlauten umgehen. Dies führt dazu, dass an den Stellen in der PNML-File, wo ein Umlaut stehen sollte, ein Fragezeichen steht. Der Algorithmus funktioniert weiterhin und die PNML-File kann auch dargestellt werden, trotzdem sollten in Zukunft die Umlaute vor Ausführung des Algorithmus entsprechend ersetzt werden, beispielsweise ein Ä durch Ae.

Ausblick

Zusammenfassend lässt sich sagen, dass Process-Mining ein interessantes und recht neues Forschunggebiet ist. Der Großteil der Forschungen im Bereich Process-Mining haben sich mit vergangenheitsorientierten Daten beschäftigt und diese Vergangenheitsicht ausführlich behandelt.

Für die Zukunft von Process-Mining klingt vor allem die Kollaboration mit einem weiteren aktuellen Trend, *Artificial Intelligence (AI)*, vielversprechend. Die Kernidee besteht darin, ein Modell mit den gesammelten Event-Daten zu trainieren, um anschließend Aussagen über die Zukunft eines Cases machen zu können. Diese Anwendung von Process-Mining fällt unter dem Begriff *Operational Support* und bietet die Möglichkeit zur Laufzeit eines Prozesses Abweichungen zu erkennen, den erwarteten Fertigstellungszeitpunkt zu berechnen oder Empfehlungen zu geben, welche Aktivität als nächstes ausgeführt werden sollte (vgl. Aalst 2016, S.305-306).

Ein weiterer wichtiger Punkt, bei dem in Zukunft weiter geforscht werden sollte, ist die Anwendung von Process-Mining auf Prozesse, die sich über System- und/oder Unternehmensgrenzen erstrecken. Hierbei geht es um die Problematik, dass Events eines Cases zusammengeführt werden müssen, obwohl keine gemeinsame CaseID vorhanden ist.

Literaturverzeichnis

[ISO20510 2011] System und Software-Engineering – Qualitätskriterien und Bewertung von System und Softwareprodukten / International Organization for Standardization. März 2011. – Standard

[Aalst u. a. 2004] Aalst, Wil M. P. ; Weijters, T. ; Maruster, L.: Workflow mining: discovering process models from event logs. In: *IEEE Transactions on Knowledge and Data Engineering* 16 (2004), Sep., Nr. 9, S. 1128–1142. – ISSN 1041-4347

[Aalst u. a. 2012] Aalst, Wil M. P. van d. ; Adriansyah, Arya ; Medeiros, Ana Karla A. de ; Arcieri, Franco ; Baier, Thomas ; Blickle, Tobias ; Bose, Jagadeesh C. ; Brand, Peter van den ; Brandtjen, Ronald ; Buijs, Joos ; Burattin, Andrea ; Carmona, Josep ; Castellanos, Malu ; Claes, Jan ; Cook, Jonathan ; Costantini, Nicola ; Curbera, Francisco ; Damiani, Ernesto ; Leoni, Massimiliano de ; Delias, Pavlos ; Dongen, Boudewijn F. van ; Dumas, Marlon ; Dustdar, Schahram ; Fahland, Dirk ; Ferreira, Diogo R. ; Gaaloul, Walid ; Geffen, Frank van ; Goel, Sukriti ; Günther, Christian ; Guzzo, Antonella ; Harmon, Paul ; Hofstede, Arthur ter ; Hoogland, John ; Ingvaldsen, Jon E. ; Kato, Koki ; Kuhn, Rudolf ; Kumar, Akhil ; La Rosa, Marcello ; Maggi, Fabrizio ; Malerba, Donato ; Mans, Ronny S. ; Manuel, Alberto ; McCreesh, Martin ; Mello, Paola ; Mendling, Jan ; Montali, Marco ; Motahari-Nezhad, Hamid R. ; Muehlen, Michael zur ; Munoz-Gama, Jorge ; Pontieri, Luigi ; Ribeiro, Joel ; Rozinat, Anne ; Seguel Pérez, Hugo ; Seguel Pérez, Ricardo ; Sepúlveda, Marcos ; Sinur, Jim ; Soffer, Pnina ; Song, Minseok ; Sperduti, Alessandro ; Stilo, Giovanni ; Stoel, Casper ; Swenson, Keith ; Talamo, Maurizio ; Tan, Wei ; Turner, Chris ; Vanthienen, Jan ; Varvaressos, George ; Verbeek, Eric ; Verdonk, Marc ; Vigo, Roberto ; Wang, Jianmin ; Weber, Barbara ; Weidlich, Matthias ; Weijters, Ton ; Wen, Lijie ; Westergaard, Michael ; Wynn, Moe: Process Mining Manifesto. In: Daniel, Florian (Hrsg.) ; Barkaoui, Kamel (Hrsg.) ; Dustdar, Schahram (Hrsg.): *Business Process Management Workshops : BPM 2011*

International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, S. 169–194. – URL https://link.springer.com/book/10.1007/978-3-642-28108-2. – Zugriffsdatum: 2018-11-12. – ISBN 978-3-642-28108-2

[Aalst 2014] Aalst, Wil M. P. van d.: *Data Scientist: The Engineer of the Future.* S. 13–26, 01 2014. – ISBN 978-3-319-04947-2

[Aalst 2015] Aalst, Wil M. P. van d.: *Extracting Event Data from Databases to Unleash Process Mining.* 2015. – URL http://wwwis.win.tue.nl/~wvdaalst/publications/p817.pdf. – Zugriffsdatum: 2019-04-27

[Aalst 2016] Aalst, Wil M. P. van d.: *Process Mining - Data Science in Action.* Berlin, Heidelberg : Springer, 2016. – ISBN 978-3-662-49851-4

[Bose u. a. 2013] Bose, R. P. Jagadeesh C. ; Mans, R. S. ; Aalst, Wil M. P. van der: Wanna improve process mining results? : it's high time we consider data quality issues seriously, 2013

[Bray 2017] Bray, T.: The JavaScript Object Notation (JSON) Data Interchange Format / RFC Editor. RFC Editor, December 2017 (90). – STD. – ISSN 2070-1721

[Günther 2009] Günther, Christian W.: *XES Standard Definition.* 2009. – URL http://www.xes-standard.org/_media/xes/xes_standard_proposal.pdf. – Zugriffsdatum: 2019-04-26

[Günther und Verbeek 2018] Günther, Christian W. ; Verbeek, Eric: *OpenXES Developer Guide.* 2018. – URL http://www.xes-standard.org/_media/openxes/openxesdeveloperguide-2.0.pdf. – Zugriffsdatum: 2019-04-15

[Hilbert und López 2011] Hilbert, Martin ; López, Priscila: The World's Technological Capacity to Store, Communicate, and Compute Information. In: *Science (New York, N.Y.)* 332 (2011), 02, S. 60–5

[Hillah und Kindler 2009] Hillah, Lom ; Kindler, Ekkart: *The Petri Net Markup Language theory and practice.* 2009. – URL http://www.pnml.org/papers/PNML-Tutorial.pdf. – Zugriffsdatum: 2019-04-24

[Kops 2018] Kops, Micha: *Qualität, funktionale und nichtfunktionale Anforderungen in der Software-Entwicklung.* 2018. – URL https://blog.seibert-media.net/blog/2018/05/14/

qualitaet-funktionale-und-nichtfunktionale-anforderungen-in-der-software-en – Zugriffsdatum: 2019-05-27

[Manyika u. a. 2011] Manyika, James ; Chui, Michael ; Brown, Brad ; Bughin, Jacques ; Dobbs, Richard ; Roxburgh, Charles ; Hung Byers, Angela: *Big data: The next frontier for innovation, competition, and productivity*. 05 2011

[de Medeiros u. a. 2003] Medeiros, A. K. A. de ; Aalst, W. M. P. van der ; Weijters, A. J. M. M.: Workflow Mining: Current Status and Future Directions. In: Meersman, Robert (Hrsg.) ; Tari, Zahir (Hrsg.) ; Schmidt, Douglas C. (Hrsg.): *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2003, S. 389–406. – ISBN 978-3-540-39964-3

[de Medeiros u. a. 2007] Medeiros, A. K. A. de ; Weijters, A. J. M. M. ; Aalst, W. M. P. van der: Genetic process mining: an experimental evaluation. In: *Data Mining and Knowledge Discovery* 14 (2007), Apr, Nr. 2, S. 245–304. – URL https://doi.org/10.1007/s10618-006-0061-7. – ISSN 1573-756X

[Reisig 2010] Reisig, Wolfgang: *Petrinetze - Modellierungstechnik, Analysemethoden, Fallstudien*. 1. Aufl. Berlin Heidelberg New York : Springer-Verlag, 2010. – ISBN 978-3-834-89708-4

[Rozinat und Aalst 2006] Rozinat, A. ; Aalst, W.M.P.: *Decision mining in business processes*. Technische Universiteit Eindhoven, 2006 (BETA publicatie : working papers). – ISBN 90-386-0685-0

[Rozinat und Aalst 2008] Rozinat, A. ; Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. In: *Information Systems* 33 (2008), Nr. 1, S. 64 – 95. – URL http://www.sciencedirect.com/science/article/pii/S030643790700049X. – ISSN 0306-4379

[Song und Aalst 2008] Song, Minseok ; Aalst, Wil M.: Towards comprehensive support for organizational mining. In: *Decision Support Systems* 46 (2008), Nr. 1, S. 300 – 317. – URL http://www.sciencedirect.com/science/article/pii/S0167923608001280. – ISSN 0167-9236

[Stäps und Blanck 2019] Stäps, Florian ; Blanck, Ilona: *Tutorial: Getting Started with the ICC*. 2019. – URL https://userdoc.informatik.haw-hamburg.de/lib/exe/fetch.php?media=docu:icc_tutorial_hello_world.pdf. – Zugriffsdatum: 2019-05-17

[Weijters und Aalst 2003] Weijters, A.J.M.M. ; Aalst, W.M.P.: Rediscovering workflow models from event-based data using Little Thumb. In: *Integrated Computer-Aided Engineering* 10 (2003), Nr. 2, S. 151–162. – ISSN 1069-2509

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort	Datum	Unterschrift im Original