UNIVERSITÄT BAYREUTH

MASTERARBEIT

Process Mining auf Basis expliziter Semantikdefinitionen

Stefan Schönig

DATENBANKEN UND INFORMATIONSSYSTEME

Lehrstuhl für Datenbanken und Informationssysteme

Angewandte Informatik IV
Universität Bayreuth

Master-Thesis im Studiengang Angewandte Informatik

Process Mining auf Basis expliziter Semantikdefinitionen

vorgelegt von

Stefan Schönig

geboren am 04.07.1986 in Weiden i. d. OPf.
st.schoenig@gmail.com

eingereicht am 24.10.2011

Lehrstuhl für Angewandte Informatik IV
Datenbanken und Informationssysteme
Universität Bayreuth

Prüfer:
Prof. Dr.-Ing. Stefan Jablonski
Prof. Dr. Thorsten Eymann

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Weiden, den 24.10.2011 _____

Kurzbeschreibung

Process Mining auf Basis expliziter Semantikdefinitionen

Business Process Management wird als wichtige Strategie angesehen um Geschäftsprozesse zu kontrollieren und zu verbessern [1]. Eine zentrale Phase dabei ist das Modellieren der Prozesse, was im Allgemeinen sehr kostenintensiv und zeitaufwändig ist. Aus diesen Gründen ist man stets auf der Suche nach neuen Methoden, welche diesen Arbeitsaufwand verringern und zur Verbesserung der Informationsversorgung beitragen. Eine Sammlung derartiger Methoden ist Process Mining. Process Mining Verfahren nutzen aufgezeichnete Prozessdaten bereits abgelaufener Prozesse um daraus Information zu extrahieren und Prozessmodelle zu generieren [2]. Diese (halb-) automatische Generierung von Prozessmodellen bietet viel Potential um Zeit und Kosten einzusparen. Es können zahlreiche Informationen bei der Ausführung von Prozessen protokolliert werden, neben der eigentlichen Reihenfolge der Prozessschritte beispielsweise auch die ausführenden Agenten, verwendete und produzierte Daten oder benutzte Werkzeuge. Aus diesem Grund wurden Process Mining Verfahren entwickelt, welche ihren Fokus jeweils auf eine bestimmte Art von Information (Perspektive des Prozesses) richten. All diesen Verfahren ist gemeinsam, dass die Möglichkeiten der zu Grunde liegenden Algorithmen nur auf deren jeweilige Aufgabe beschränkt sind und diese nicht flexibel auf neue Anforderungen oder neu gestellte Aufgaben anpassbar sind. Außerdem erfolgt keine kombinierte Betrachtung der verschiedenen Arten von Information. Gerade derartige Zusammenhänge, welche perspektiven-übergreifend existieren, sind jedoch für die Analyse von Geschäftsprozessen interessant und wichtig.

Process Mining auf Basis expliziter Semantikdefinitionen überführt die aufgezeichneten Prozessdaten in eine allgemeingültige Graph-Datenstruktur, welche es ermöglicht, flexibel nach beliebig definierten Semantiken zu suchen. Dadurch kann der Aufgabenbereich des Verfahrens flexibel ausgeweitet oder verkleinert werden. Außerdem wird eine kombinierte Betrachtung der vorhandenen Information ermöglicht. Das vorgestellte Verfahren ermöglicht daher die Erkennung von Zusammenhängen zwischen den verschiedenen Arten von gespeicherten Prozessdaten.

Abstract

Process mining based on explicit semantic definitions

Business process management is considered as an essential strategy to create and maintain competitive advantage by modeling, controlling and monitoring production and development as well as administrative processes [1]. Business process management starts with a modeling phase which is very time and cost intensive. Since process modeling is a very expensive and cumbersome task, approaches are identified that reduces the modeling effort. One of them is process mining. Process mining utilizes information/knowledge about processes whilst execution [2]. The fundamental idea is to extract knowledge from event logs recorded by information systems. Thus, process mining aims at the (semi-) automatic reconstruction of process models using information provided by event logs. There are several types of information that can be recorded within a log. In addition to the sequence of process steps, it's possible to record the performing agents, corresponding data or tools. For this reason, process mining algorithms have been developed, focusing different types (perspectives of the process) of information. However, the possibilities of all approaches are limited to their predefined tasks. There is no possibility to react to new defined demands flexibly. Furthermore, there is no combined examination of the different information types. However, relations between the different perspectives are especially interesting and important for the analysis of business processes.

Therefore, process mining based on explicit semantic definitions converts the recorded process information into a data structure that offers the possibility to search for user-defined semantics and rules. Hence, the scope of functions can be extended or reduced flexibly. Moreover, the approach offers the possibility to examine the recorded process information comprehensively. This way, relations between the different perspectives of the process can be discovered.

Inhaltsverzeichnis

1 Einführung ... 1
1.1 Motivation ... 1
1.2 Aufbau und Ziel dieser Arbeit ... 1
2 Grundlagen ... 3
2.1 Begriffe aus dem Business Process Management ... 3
2.2 Perspektivenorientierte Prozessmodellierung (POPM) ... 4
2.3 Wissensrepräsentation und Wissensverarbeitung ... 5
2.3.1 Process Execution Log als Wissensbasis ... 5
2.3.2 Ontologien und Beschreibungssprachen ... 5
2.3.3 Open- und Closed-World Annahme ... 7
3 Related Work ... 9
3.1 Process Mining ... 9
3.1.1 Übersicht ... 9
3.1.2 Process Mining auf Basis gerichteter Graphen ... 10
3.1.3 Kontrollfluss-Verfahren nach van der Aalst et al. ... 11
3.1.4 Betrachtung der organisatorischen Perspektive ... 20
3.1.5 Verfahren der datenorientierten Perspektive ... 21
3.2 Semantikdefinitionen in der Prozessausführung ... 22
4 Process Mining auf Basis expliziter Semantikdefinitionen ... 25
4.1 Zieldefinition und Annahmen ... 26
4.1.1 Ziele ... 26
4.1.2 Voraussetzungen und Einschränkungen ... 27
4.2 Grundlegende Analyse des Process Execution Logs ... 27
4.2.1 Klassifikation von Parallelitäten und Nachfolgebeziehungen ... 28
4.2.2 Aufbau von Instanzgraphen ... 30
4.3 Semantische Interpretation der Instanzgraphen ... 32
4.3.1 Algorithmische Umsetzung der semantischen Interpretation ... 32
4.3.2 Generierung des Semantikgraphen ... 36
4.3.3 Kapselung von Prozessen ... 37
4.4 Generierung von Empfehlungen ... 41
4.4.1 Bestimmung von Zusammenhangskomponenten ... 41
4.4.2 Überprüfung einer Recommendation-Bedingung ... 42
4.5 Zusammenfassung der extrahierten Information ... 43
5 Exemplarische Implementierung auf Basis ausgewählter Semantikdefinitionen ... 45
5.1 Grundlegende Analyse des Logs ... 45
5.1.1 Parsen der JSON Logdatei ... 45
5.1.2 Aufbau instanzspezifischer Eventlisten ... 46
5.2 Erzeugung und Implementierung der Instanzgraphen ... 47
5.3 Implementierung der semantischen Interpretation ... 48
5.3.1 Grundlegende Vorgehensweise und Architektur ... 48
5.3.2 Exemplarische Implementierungen des Interfaces IEdgeType ... 50
5.3.3 Implementierung der Prozesskapselung ... 52
5.4 Implementierung von Empfehlungen ... 54
5.5 Komplexitätsbetrachtung ... 55
5.6 Überführung des Semantikgraphen in ein Prozessmodell ... 56
6 Zusammenfassung und Ausblick ... 59
Literaturverzeichnis ... 61

Abbildungsverzeichnis

Abbildung 1: Vereinfachtes Modell des BPM Lifecycle nach [1] 3
Abbildung 2: Abbildungsregeln des α-Algorithmus (vgl. [5]) 13
Abbildung 3: Konstruiertes Prozessmodell des α-Algorithmus (vgl. [5]) 14
Abbildung 4: Abhängigkeitsgraph des Heuristic-Miner Algorithmus (vgl. [24]) 16
Abbildung 5: Beispiel eines Soziogramms "handover of work" (vgl. [26]) 21
Abbildung 6: Semantikdefinitionen liefern die Bedeutung von Modellierungselementen [28] 23
Abbildung 7: Validierung von ausgeführten Prozessen [28] 23
Abbildung 8: Process Mining auf Basis von Semantikdefinitionen 25
Abbildung 9: Aufteilung der Logeinträge nach Prozessinstanzen 29
Abbildung 10: Verschiedene Arten von Parallelitäts- und Nachfolgerbeziehungen 29
Abbildung 11: Gemeinsamer Graph für alle Instanzen 30
Abbildung 12: Separate Graphen für jede Prozessinstanz (Instanzgraphen) 31
Abbildung 13: Aufbau einer Semantikdefinition 32
Abbildung 14: Algorithmus zum Beweis der *BossConnector* Semantik 35
Abbildung 15: Verschiedene Arten von Bedingungen erzeugen unterschiedliche Constraints 36
Abbildung 16: Zusammenfassende Vorgehensweise zur Generierung des Semantikgraphen 37
Abbildung 17: Exemplarischer Semantikgraph zur Generierung von Prozesshierarchien 39
Abbildung 18: Erkennung von Kapseln innerhalb des Semantikgraphs 40
Abbildung 19: Resultierende Prozesshierarchien 40
Abbildung 20: Exemplarische Darstellung der kompositen Prozesse als Prozessmodell 40
Abbildung 21: Teilgraphen zu jedem kompositen Prozess (Kapsel) 42
Abbildung 22: Resultierende Zusammenhangskomponenten anhand des Algorithmus von [31] 42
Abbildung 23: Exemplarisches Klassendiagramm der semantischen Interpretation 49
Abbildung 24: Resultierendes Prozessmodell auf Basis eines exemplarischen Meta-Modells 57
Abbildung 25: Vollständige Integration in ein Meta-Modellierungsframework 60

Tabellenverzeichnis

Tabelle 1: Ausschnitt eines Process Execution Log als Wissensbasis 5
Tabelle 2: Fähigkeiten und Grenzen des α-Algorithmus (in Anlehnung an [20]) 15
Tabelle 3: Beispiel einer Wertematrix des Heuristic-Miner Algorithmus (vgl. [24]) 16
Tabelle 4: Kausalitätsmatrix des Heuristic-Miner Algorithmus (vgl. [24]) 17
Tabelle 5: Zusammenfassung des Heuristic-Miner Algorithmus (in Anlehnung an [20]) 17
Tabelle 6: Zusammenfassung des Genetischen Process Mining (in Anlehnung an [20]) 19
Tabelle 7: Überblick über verschiedene Algorithmen (in Anlehnung an [20]) 20
Tabelle 8: Beispiel eines Logs mit zu Grunde liegenden Prozessdaten 28

1 Einführung

1.1 Motivation

Business Process Management wird als wichtige Strategie angesehen um Geschäftsprozesse zu kontrollieren und zu verbessern [1]. Viele Organisationen nutzen einen prozessbasierten Ansatz um ihre Aktivitäten zu steuern. Eine zentrale Phase dabei ist das Modellieren der Prozesse, was im Allgemeinen sehr kostenintensiv und zeitaufwändig ist. Die Modellierung der Geschäftsprozesse eines Unternehmens oder einer Institution bedarf tiefgreifendes Fachwissen der zu Grunde liegenden Anwendung und umfangreiche Diskussionen mit beteiligten Fachexperten, um alle verschiedenen Bereiche des Prozesses beleuchten zu können [2]. Aus diesen Gründen ist man stets auf der Suche nach neuen Methoden, welche diesen Arbeitsaufwand verringern und zur Verbesserung der Informationsversorgung beitragen. Eine Sammlung derartiger Methoden ist Process Mining. Diese Verfahren nutzen aufgezeichnete Prozessdaten bereits abgelaufener Prozesse um daraus Information zu extrahieren und Prozessmodelle zu generieren [2]. Diese (halb-) automatische Generierung von Prozessmodellen bietet viel Potential um Zeit und Kosten einzusparen. Es können zahlreiche Informationen bei der Ausführung von Prozessen protokolliert werden. Neben der eigentlichen Reihenfolge der Prozessschritte auch die ausführenden Agenten, verwendete und produzierte Daten oder benutzte Werkzeuge. Aus diesem Grund wurden Mining Verfahren entwickelt, welche ihren Fokus jeweils auf eine bestimmte Art von Information (Perspektive des Prozesses) richten: Kontrollfluss-Verfahren ermitteln die Reihenfolge von Prozessen, deren Abhängigkeiten und zeigen eine Möglichkeit zur parallelen Ausführung auf. Verfahren, welche die gespeicherten Agenten betrachten, konstruieren soziale Netze. Diese ermöglichen einen Einblick in die Organisationsstrukturen eines Unternehmens. Es kann daraus extrahiert werden, welche Mitarbeiter zu welchem Grad miteinander zusammengearbeitet haben. All diesen Verfahren ist jedoch gemeinsam, dass die Möglichkeiten der zu Grunde liegenden Algorithmen nur auf deren jeweilige Aufgabe beschränkt sind und diese nicht flexibel auf neue Anforderungen oder neu gestellte Aufgaben anpassbar sind. Außerdem erfolgt keine kombinierte Betrachtung der verschiedenen Arten von Information: Es wird *entweder* der Kontrollfluss *oder* die Organisationsstruktur betrachtet, jedoch nicht der Kontrollfluss *in Abhängigkeit von* der Organisationsstruktur. Gerade derartige Zusammenhänge sind jedoch für die Analyse von Geschäftsprozessen interessant und wichtig. Durch eine kombinierte Betrachtung der verschiedenen Perspektiven wird die Beantwortung der Frage nach dem Grund für eine Ausführungsreihenfolge, einer bestimmten Agentenzuordnung und zahlreicher anderer Fragen ermöglicht.

1.2 Aufbau und Ziel dieser Arbeit

Angesichts der eben beschriebenen Motivation wird in dieser Arbeit ein neues Process Mining Verfahren vorgestellt. Ziel dieses Verfahrens ist es, die aufgezeichneten Prozessdaten in eine allgemeingültige Datenstruktur zu überführen, welche es ermöglicht, flexibel nach beliebig definierten Regeln zu suchen. Dadurch kann der Aufgabenbereich des Algorithmus flexibel ausgeweitet oder verkleinert werden. Ein weiteres Ziel dieser Arbeit ist die kombinierte Betrachtung der vorhandenen Information. Das vorgestellte Verfahren soll Zusammenhänge zwischen verschiedenen Arten von gespeicherten Prozessdaten erkennen können. Neben diesen verschiedenen Arten von Prozessdaten werden in Kapitel 2 wichtige Grundlagen aus dem Business Process Management und der digitalen Wissensrepräsentation eingeführt, welche für das weitere Verständnis unverzichtbar sind. Da das Forschungsgebiet des Process Mining nicht neu ist und

bereits einige Verfahren existieren, verschafft Kapitel 3 einen Überblick über die wichtigsten bestehenden Algorithmen und klassifiziert deren Möglichkeiten nach ausgewählten Bewertungskriterien. Da das in dieser Arbeit vorgestellte Verfahren auf expliziten Semantikdefinitionen basiert, werden in Kapitel 3 ebenso grundlegende Forschungsentwicklungen in Bezug auf Semantikdefinitionen in der Prozessausführung dargelegt. In Kapitel 4 wird schließlich die Vorgehensweise des neuen Process Mining Ansatzes *SemanticPM* aufgezeigt und anhand von einigen Beispielen erläutert. In Kapitel 5 wird der Ansatz anhand von einigen, exemplarisch ausgewählten Semantikdefinitionen implementiert. Anhand von Code-Ausschnitten kann dieses Kapitel als Anleitung zur Implementierung beliebiger weiterer Semantiken dienen. Kapitel 6 fasst die Ergebnisse schließlich zusammen und gibt einen Ausblick auf zukünftige Forschungs- und Entwicklungsmöglichkeiten in diesem Bereich.

2 Grundlagen

2.1 Begriffe aus dem Business Process Management

Geschäftsprozessmanagement (engl. business process management, BPM) beschäftigt sich mit der Identifikation, Gestaltung, Dokumentation, Implementierung, Steuerung und Verbesserung von Geschäftsprozessen [3]. Der Begriff *Prozess* orientiert sich in dieser Arbeit an der Terminologie von [4]. Prozesse sind Vorgänge innerhalb einer Organisation, die die Erfüllung einer bestimmten Aufgabe zum Ziel haben. Im Allgemeinen sind sie grobgranular und damit nicht ausführbar. Viele Organisationen verwenden einen prozessbasierten Ansatz zur Verwaltung und Steuerung von Abläufen. Die einzelnen Bereiche des BPMs werden im sog. *BPM Lifecycle* (siehe Abbildung 1) näher beschrieben. Der traditionelle BPM Lifecycle besteht aus den folgenden Phasen [1]: Prozessdesign und Prozessmodellierung, Prozessimplementierung, Prozessausführung und Prozessaufzeichnung sowie der Prozessevaluation (u.a. auf Basis der Ergebnisse durch Process Mining). Die Ergebnisse der Evaluation werden schließlich verwendet um den bestehenden Prozess zu verbessern.

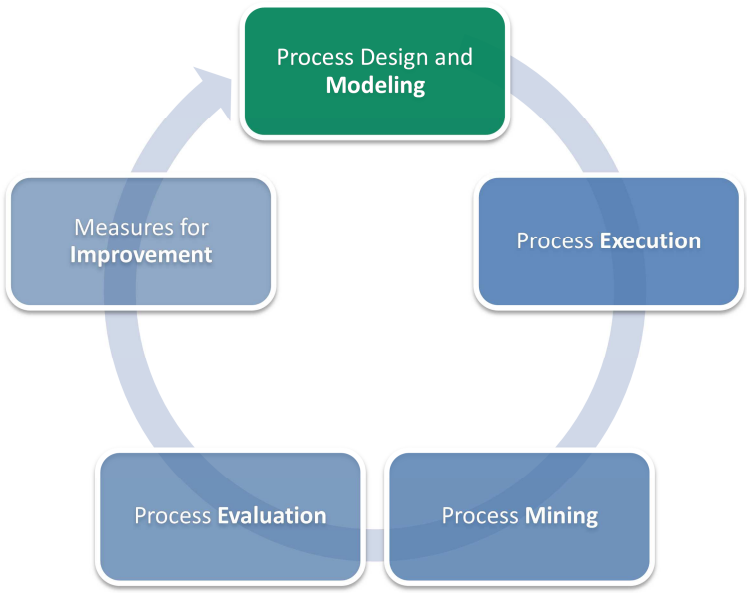

Abbildung 1: Vereinfachtes Modell des BPM Lifecycle nach [1]

Prozessmodellierung

Business process management beginnt mit der Modellierungsphase. Bei der Geschäftsprozessmodellierung (engl: business process modeling) werden Geschäftsprozesse meist grafisch dargestellt und somit modelliert. Der Schwerpunkt liegt auf dem Darstellen des Ablaufs, aber auch Daten und Organisation werden abgebildet. Diese Phase kann unter Umständen sehr zeit- und kostenaufwendig sein. Sie erfordert tiefgehende Kenntnisse der zu Grunde liegenden Abläufe und detaillierte Gespräche mit involvierten Mitarbeitern und Experten [2]. Zur Erreichung einer möglichst hohen Flexibilität bei Prozessmodellen wurde die perspektivenorientierte Prozessmodellierung entwickelt. Diese wird in Abschnitt 2.2 detaillierter erläutert.

Prozessausführung und Monitoring

In der Implementierungsphase werden die Prozessmodelle in Ausführungsumgebungen transferiert und implementiert. Prozessausführung kann einerseits manuell erfolgen (z. B. anhand eines Ablaufhandbuchs) anderseits auch automatisiert und informationstechnisch unterstützt durch ein *Workflow-Management-System (WfMS)* [1]. *Workflows* stellen durch ihre spezifische und detailgetreue Modellierung ausführbare Prozesse dar. Durch Anreicherung eines Geschäftsprozesses mit genügend Informationen kann dieser somit als Workflow auf einem WfMS zur Ausführung gebracht werden [4]. Der Vorgang der Aufzeichnung von Informationen (z.B. Start-Stop-Ereignisse von Prozessschritten, beteiligte Personen, verwendete Werkzeuge, usw.) während der Prozessausführung wird als *Process Execution Monitoring* bzw. *Process Execution Logging* bezeichnet.

Process Mining und Prozessevaluation

In der Evaluationsphase des BPM Lifecycle nutzt man u. a. die gespeicherten Daten laufender und abgelaufener Prozessausführungen um Prozesse zu kontrollieren und zu verbessern. Eine Sammlung an Methoden und Werkzeugen welche Prozessausführungsdaten (Process Execution Log, kurz: Log) verwenden ist *Process Mining*. Die Idee des Process Mining ist es, Wissen aus Ereignis-Logs zu extrahieren. Process Mining Techniken und Werkzeuge ermöglichen das Auffinden von Prozess-, Kontroll-, Daten- und Organisationsstrukturen innerhalb eines Logs [5]. Ziel des Process Mining ist damit die Konstruktion eines dem aufgezeichneten Prozess zu Grunde liegenden Prozessmodells.

2.2 Perspektivenorientierte Prozessmodellierung (POPM)

Bei diesem in [4] eingeführten Modell wird ein Prozess durch verschiedene Perspektiven repräsentiert, die je nach Bedarf erweiterbar sind. Die verschiedenen Aspekte dieses Modellierungsansatzes werden nachfolgend detaillierter beschrieben.

Funktionale Perspektive

Die Funktionale Perspektive deklariert die funktionalen Einheiten. Unterschieden wird zwischen elementaren Prozessen, die keine weiteren Subprozesse beinhalten und kompositen Prozessen, die sich wiederum aus anderen Prozessen – sowohl elementar als auch komposit – zusammensetzen können.

Verhaltensbezogene Perspektive

Die Verhaltensbezogene Perspektive legt den Kontrollfluss, d.h. eine mögliche Ausführungsreihenfolge der funktionalen Einheiten fest. Denkbar sind dabei eine strikte und komplett vorgegebene Reihenfolge bis hin zu einer völlig freien Auswahl des Ablaufs. Der Kontrollfluss kann z.B. sequentiell, unabhängig oder zyklisch verlaufen.

Datenbezogene Perspektive

Die datenorientierte Perspektive beschreibt den Aufbau aller vom Prozess verwendeten Daten. Hier wird festgelegt, welche Eingangsdaten ein Prozess verwendet und welche Ausgangsdaten er produziert. Des Weiteren wird der Datenflusses innerhalb des Prozesses spezifiziert, d.h. auf welchem Weg gelangen die Daten vom Eingang zum Ausgang und wie werden sie dabei verarbeitet.

Organisatorische Perspektive

Mit Hilfe der Organisatorischen Perspektive wird festgelegt, welche Person für die Ausführung eines Prozesses verantwortlich ist. Denkbar ist u. a. eine Rechteverwaltung, die einzelnen Benutzern oder Rollen entsprechende Ausführungsrechte zuteilt.

Operationale Perspektive
Die operationale Perspektive spezifiziert Werkzeuge und Programme, welche für die Ausführung der jeweiligen funktionalen Einheit verwendet werden. Das kann beispielsweise Microsoft Excel als Tabellenkalkulation oder Adobe Photoshop für die Grafikbearbeitung sein.

2.3 Wissensrepräsentation und Wissensverarbeitung

2.3.1 Process Execution Log als Wissensbasis

Wissensrepräsentation (englisch: *knowledge representation*) dient im Rahmen der Wissensmodellierung dazu, Wissen in Systemen formal abzubilden [6]. Dazu sind verschiedene formale Sprachen entwickelt worden. Auf diese Weise repräsentiertes Wissen wird als *Wissensbasis* bezeichnet. Die verwendete Wissensbasis als Grundlage für Process Mining und somit auch dieser Arbeit wird von einem Process Execution Log bereitgestellt. Ein Element der darin enthaltenen Wissensbasis kann dabei aus einem (beliebigem) n-Tupel an Information pro aufgezeichnetem Prozess(-ereignis) bestehen. Nach [7] werden einige Anforderungen an die Inhalte von Logs gestellt, um ein erfolgreiches Analysieren mit Hilfe von Process Mining Techniken zu ermöglichen:

- Jeder Eintrag in der Ereignis-Logdatei bezieht sich auf genau eine Aktivität (d.h. auf ein eindeutig identifizierbares Ereignis eines Prozessschrittes).
- Jeder Eintrag bezieht sich auf genau eine Prozessinstanz.
- Die Einträge besitzen einen Zeitstempel (timestamp) und unterliegen einer Totalordnung.

Neben diesen Informationen können jedoch beliebig viele weitere Prozessdaten im Log enthalten sein. Diese Daten lassen sich im Allgemeinen einer der Perspektiven des POPM zuordnen. Tabelle 1 zeigt beispielhaft ein Process Execution Log welches als Wissensbasis für die Verfahren des Process Mining dienen könnte. Neben eindeutigen Ereignis- und Prozessbezeichnern und einem Timestamp ist jedem Tupel des Logs eine eindeutige Prozessinstanz (d.h. ein bestimmter Fall) zugeordnet. Der *Action Type* gibt an, ob das jeweilige Ereignis den Start oder das Ende eines Prozessschrittes markiert. Die Spalten Agents (organisationsorientiert), Data (datenorientiert) und Tools (operationsorientiert) sind zusätzliche Informationen, welche bei Ausführung des Prozesses gespeichert wurden.

Event ID	Process ID	Case/Instance	Action Type	Agents	Data	Tools	Timestamp
1	**A**	**1**	start	Jack	…	…	…
2	**A**	**1**	finish				
3	**C**	*2*	start				
4	**C**	*2*	finish				
5	**B**	**1**	start				

Tabelle 1: Ausschnitt eines Process Execution Log als Wissensbasis

2.3.2 Ontologien und Beschreibungssprachen

Unter Umständen sind umfangreiche Detailinformationen zu gespeicherten Prozessdaten vorhanden, welche ebenso nützlich sein können um im Rahmen der Prozessevaluation und des Process Mining Einflüsse auf die Prozessausführung ermitteln zu können. Beispielsweise ist neben den aufgezeichneten prozessverantwortlichen Personen zusätzlich eine Organisationsstruktur gegeben. Diese lässt Rückschlüsse dahingehend zu, welche Prozesse von Führungspersonen und welche Prozesse von einfachen Mitarbeitern ausgeführt werden müssen.

Um nicht sämtliche Informationen eines Wissensgebietes in jedes Tupel eines Logs speichern zu müssen, bedient man sich u. a. Ontologien [8]. In der Informatik beschreibt der Begriff *Ontologie* eine Repräsentation der grundlegenden Begriffe und Zusammenhänge eines Wissensgebiets. Dort werden Begriffe und ihre Zusammenhänge definiert. Zur Darstellung wird eine formale Repräsentation, eine sog. Ontologiebeschreibungssprache verwendet. Eine vollständige Übersicht und Beschreibung aller Ontologiebeschreibungssprachen würde den Rahmen dieser Arbeit überschreiten, weshalb im Folgenden nur die wichtigsten Beschreibungssprachen kurz erläutert werden und auf detailliertere Referenzen verwiesen wird.

RDF (Resource Description Framework) und RDF Schema

Das *Resource Description Framework (RDF)* ist eine formale Sprache um strukturierte Informationen zu beschreiben. Durch RDF sollen Anwendungen in die Lage versetzt werden, Daten im Web auszutauschen, ohne dass ihre ursprüngliche Bedeutung dabei verloren geht [8]. RDF ermöglicht durch dessen Modellierung eine maschinenseitige Verarbeitung der Informationen. Das RDF-Modell bietet eine syntaxunabhängige Darstellungsform für Ausdrücke und besteht aus drei Elementen:

- **Ressourcen:** Alle Dinge, die durch RDF-Ausdrücke beschrieben werden sind Ressourcen. Die eindeutige Kennzeichnung erfolgt dabei über URI (Uniform Resource Identifier)
- **Eigenschaften:** Eine Ressource wird durch ihre Eigenschaften beschrieben. Diese Eigenschaften können durch weitere Beschreibungen genauer spezifiziert werden.
- **Aussagen:** Eine Aussage besteht aus einer Ressource, einer Eigenschaft und einem entsprechendem Wert.

Aussagen werden als Sätze der Form Subjekt, Prädikat, Objekt interpretiert, wobei die Ressource als Subjekt, die Eigenschaft als Prädikat und der Wert der Eigenschaft als Objekt zu verstehen sind [9]. RDF kann auf drei verschiedene Weisen dargestellt werden: Grafisch, als N-Tripel oder in Form von XML.

Die Erweiterung *RDF Schema* ist nichts anderes als ein spezielles RDF-Vokabular. Jedes RDFS-Dokument ein syntaktisch korrektes RDF-Dokument [8]. Grundlegend ermöglicht es RDF Schema Hintergrundinformationen - sogenanntes *terminologisches Wissen* – über die im RDF Dokument verwendeten Ressourcen zu spezifizieren. Es erlaubt beispielsweise Ressourcen einer bestimmten Klasse zuzuordnen. Bezeichnungen und Eigenschaften können über RDFS nicht nur für einzelne Individuen sondern für ganze Klassen definiert werden. Damit ist es möglich neue Vokabulare zu spezifizieren, wie beispielsweise das Vokabular „FOAF“ (Friend of a friend), welches eingeführt wurde um Beziehungen zwischen Personen auszudrücken. Für weitergehende Details wird auf die W3C Spezifikation [9] verwiesen.

OWL (Web Ontology Language)

Die *Web Ontology Language (OWL)* ist eine Auszeichnungssprache zum Veröffentlichen und Austauschen von Ontologien. OWL basiert technisch auf RDF/XML, geht aber über die Ausdrucksmächtigkeit von RDF Schema hinaus. Es werden zusätzliche Sprachkonstrukte eingeführt, um die Möglichkeiten zur Beschreibung von Properties und Klassen zu erhöhen. OWL bietet die Möglichkeit, Gleichheit und Ungleichheit zwischen Klassen und Individuen, Property-Eigenschaften, wie z.B. Transitivität, Symmetrie und Kardinalitätsbeschränkungen auszudrücken. Auf Basis der Klassifizierung entsteht eine Klassenhierarchie in Form einer Baumstruktur. Diese bezeichnet man als Taxonomie. Taxonomien bezeichnen letztendlich Ontologien, welche über isSubclassOf-Properties verfügen. Eine detaillierte Referenz zu OWL liefert [10].

Mit Ontologien können Abhängigkeiten zwischen Ressourcen ausgedrückt werden, nicht aber logische Schlüsse. Um Schlussfolgerungen ziehen zu können, wird Logik benötigt. Fakten, die durch Ontologien ausgedrückt werden, können als Grundlage für logische Schlussfolgerungen verwendet werden. Ein sog. *Reasoner* kann anschließend eine Ontologie auf ihre Konsistenz hin überprüfen oder neue logische Schlüsse aus den gegebenen Fakten gewinnen.

SWRL (Semantic Web Rule Language)
Regelsprachen setzen direkt auf einer Ontologie auf. Durch den Einsatz von Regelsprachen auf einer Ontologie, kann deren Ausdrucksmächtigkeit durch neue und komplexere Konstrukte erhöht werden. Für den Einsatz auf Ontologien gibt es mehrere Regelsprachen, wobei in dieser Arbeit die *Semantic Web Rule Language (SWRL)* verwendet wird. SWRL ist eine W3C Member Submission und basiert auf einer Kombination von OWL DL/Lite mit Datalog RuleML. SWRL unterstützt drei Darstellungsformen (vgl. [11]):
XML Syntax basierend auf RuleML, OWL XML Syntax sowie eine Syntax auf einem höheren Abstraktionsniveau. Um die Übersichtlichkeit zu wahren wird in dieser Arbeit letztgenannte Syntax verwendet. Die Regeln haben folgenden Aufbau:

```
<Bedingung>   ⇨   <Konsequenz>
```

Dies bedeutet: Ist die Bedingung erfüllt, so muss die Konsequenz auch gelten. Für den Aufbau der Bedingung bzw. der Konsequenz gelten folgende Regeln:

- Sowohl die Bedingung als auch die Konsequenz können aus keinem oder mehreren Atomen bestehen.
- Ist keine Bedingung vorhanden, so gilt die Konsequenz immer.
- Eine leere Konsequenz bedeutet, dass die Bedingung ebenfalls nicht erfüllt sein darf.
- Mehrere Atome werden als Konjunktion betrachtet.
- Atome sind Ausdrücke der Form:

    1. C(x) - C ist eine OWL-Klasse
    2. P(x, y) - P ist eine OWL-Property
    3. sameAs(x, y), differentFrom(x, y) - x, y sind entweder Variablen, OWL Individuen oder konkrete OWL Daten

Beispiele für Regeln in SWRL sind:

```
   hatMutter(?x, ?y) & hatBruder(?y, ?z)   ⇨   hatOnkel(?x, ?z)
Person(?x) & hatVater(?y, ?x) & männlich(?y)   ⇨   hatSohn(?x, ?y)
```

Für weitere Details und syntaktische Einzelheiten wird auf [11] verwiesen.

2.3.3 Open- und Closed-World Annahme

Bei der Modellierung einer Wissensrepräsentation kann man von zwei unterschiedlichen Standpunkten ausgehen, der Closed World Assumption und der Open World Assumption [12]. Bei der ersten Annahme geht man davon aus, dass die verwendete Datenbasis vollständig ist und alles, was nicht explizit als wahr bekannt ist oder hergeleitet werden kann, falsch ist. Diese Annahme wird standardmäßig bei Datenbanksystemen verwendet. Betrachtet man beispielsweise die Abfrage nach einem bestimmten Artikel in der Produktdatenbank eines Unternehmens, so kann man davon ausgehen, dass falls eine leere Ergebnismenge zurückgeliefert wird, es den Artikel nicht gibt, da alle Artikel in der Produktdatenbank

aufgeführt werden. In der Beschreibungslogik gilt diese Annahme im Allgemeinen nicht. Hier gilt oft die Open-World Annahme, d.h. fehlt hier eine Information in der Wissensbasis, liefert ein entsprechender Reasoner nicht *falsch* zurück sondern *unknown*. Damit werden die Möglichkeiten von Inferenzschlüssen eingeschränkt.

Bei der Analyse und Interpretation von Process Execution Logs durch Process Mining bietet sich ebenso die Auswahl zwischen einer Closed- bzw. Open World Annahme. Ein Großteil der bestehenden Process Mining Ansätze basiert auf der Open-World Annahme. Man geht davon aus, dass die aufgezeichneten Logs evtl. unvollständig oder sogar falsch sind. Die Prozessmodelle, die durch diese Verfahren generiert werden, basieren daher beispielsweise auf Schwellenwerten auf Basis derer entschieden wird, welche Elemente in das Modell gelangen. In dieser Arbeit wird im Gegensatz dazu ein Ansatz auf Basis der Closed-World Annahme verfolgt. Eine Erläuterung für diese Annahme findet sich in Abschnitt 4.1.

3 Related Work

Nachdem im vorherigen Kapitel die Grundlagen dieser Arbeit erläutert wurden, werden in den folgenden Abschnitten Zusammenhänge mit thematisch verwandten wissenschaftlichen Arbeiten hergestellt. Da in dieser Arbeit ein neuer Process Mining Ansatz vorgestellt wird, wird vor allem auf bereits bestehende Verfahren in diesem Bereich eingegangen.

3.1 Process Mining

3.1.1 Übersicht

Die Idee der automatischen Generierung von Prozessmodellen durch die Analyse von Ereignisprotokollen wurde durch Cook und Wolf im Bereich von Software Engineering Prozessen eingeführt [13]. In den folgenden Jahren entwickelten van der Aalst et al. weitere Techniken und verwendeten diese im Bereich des Workflow Managements unter dem Begriff „Process Mining“ [14]. Im Allgemeinen ist das Ziel des Process Mining Information über Prozesse aus Ereignis-Logs von Informationssystemen zu extrahieren [7]. Es gibt zahlreiche Algorithmen und sogar komplette Tools, wie das ProM Framework [15], welche Prozessmodelle automatisch generieren. Während der letzten Jahre wurden Algorithmen entwickelt, die verschiedene Perspektiven der Prozessausführungsdaten betrachten. Van der Aalst et al. geben in [7, 14] einen detaillierten Überblick über das Thema Process Mining. Im Rahmen dieser Arbeit werden die wichtigsten Meilensteine der Entwicklungen erklärt und bewertet.

- Nach [29] können bestehende Verfahren eindeutig einer Perspektive (Kontrollfluss-, Organisatorische- oder Datenperspektive) zugeordnet werden. Eine perspektivenübergreifende Betrachtung kann somit nicht erfolgen. Lediglich im Fall der Kombination von zwei Verfahren (im Fall des Decision Mining, siehe 3.1.5) werden Zusammenhänge zwischen Perspektiven extrahiert.

- Bisherige Verfahren des Process Mining versuchen imperative Prozessmodelle zu konstruieren und basieren auf festgelegten Semantiken. Die Funktionsweise dieser Algorithmen hängt vollständig von diesen Semantiken ab. Aus diesem Grund müssen diese Algorithmen bei einer alternativen Interpretation neu entworfen und realisiert werden.

- Die Konstruktion eines vollständigen Prozessmodells mit festgelegten Semantiken stellt eine sehr schwer zu lösende Aufgabe dar. Prozessmodelle, welche komplexe Abläufe auf Basis einiger weniger Modellierungskonstrukte versuchen darzustellen, können schnell einen hohen Komplexitätsgrad erreichen.

Zuerst werden die wichtigsten Verfahren der verhaltensorientierten Perspektive vorgestellt. Diese Algorithmen haben das Ziel den Kontrollfluss des zu Grunde liegenden Prozesses aus dem Log zu rekonstruieren. Exemplarisch für ein Verfahren der organisatorischen Perspektive wird der Social Network Miner vorgestellt. Zuletzt wird das Decision Mining beleuchtet, ein Algorithmus der auf Grundlage eines Kontrollflussmodells die datenorientierte Perspektive betrachtet. Dies ist bisher das einzige Verfahren, welches Zusammenhänge zwischen (zwei) verschiedenen Perspektiven herstellt.

3.1.2 Process Mining auf Basis gerichteter Graphen

Da auch der in dieser Arbeit in Kapitel 4 vorgestellte Process Mining Ansatz auf gerichteten Graphen basiert, werden in diesem Abschnitt zuerst Verfahren beschrieben, welche ebenso Graphen als Datenstrukturen oder zur Modellierung verwenden. Es ist jedoch festzuhalten, dass es sich dabei um frühe Entwicklungen handelt und diese Verfahren in den folgenden Jahren nicht weiter verbessert wurden. Prozesse werden hier durch Knoten, kausale Beziehungen zwischen Prozessen durch Kanten modelliert. Die Aufgabenstellung liegt daher darin, einen gerichteten Graphen abzuleiten, der konsistent mit den Verlaufsdaten ist, d.h. die Log-Daten lassen sich mit dem Graphen erzeugen. Neben der Repräsentation des Prozessmodells sind sich die im Folgenden vorgestellten Ansätze auch vom prinzipiellen Ablauf her sehr ähnlich. Es wird ein Graph generiert, indem Graphen für einzelne Prozess-Anwendungsfälle (Instanzen) erzeugt werden. Diese werden anschließend nach bestimmten Kriterien zusammengefasst.

3.1.2.1 Ansatz nach Agrawal et al.

In [16] stellen Agrawal et al. einen Ansatz vor, der sich auf zyklenfreie Graphen beschränkt. Es wird ein gerichteter Graph generiert, indem für jede Nachfolgerbeziehung innerhalb des Logs eine Kante erzeugt wird. Ein Prozess A folgt einem Prozess B, falls A beginnt, nachdem B beendet ist, oder A einem Prozess C folgt, welcher auf B folgt. Die so erzeugten Kanten stellen mögliche kausale Beziehungen und damit Kanten im resultierenden Prozessgraphen dar. Da bisher jede Prozessinstanz isoliert betrachtet wurde, können im Graphen Kanten zwischen Prozessen enthalten sein, die voneinander unabhängig sind. Um diese Kanten zu entfernen werden alle Kanten entfernt, die in beiden Richtungen vorkommen oder zu einem stark zusammenhängenden Teilgraphen gehören. Da das dem Process Execution Log zu Grunde liegende Modell zyklenfrei sein soll, sind diese Zyklen ein Zeichen dafür, dass die Prozesse unabhängig voneinander sind. Anschließend werden alle Kanten entfernt, die für die Ausführung der Instanzen nicht notwendig sind. Dies erfolgt, indem für jede Instanz der induzierte Teilgraph im Graph gefunden wird und transitive Kanten in jedem Teilgraph entfernt werden. Dies ist nachzuvollziehen, da durch die transitive Definition der Nachfolgerbeziehung auch für indirekte Nachfolgerbeziehungen Kanten erzeugt wurden. Um mit Zyklen umzugehen gehen Agrawal et al. wie folgt vor: Mehrfach auftretende Prozesse werden in einem vorgelagerten Schritt durchnummeriert und wie unterschiedliche Prozesse behandelt. Nach Ausführung des Algorithmus werden die mehrfach auftretenden Prozesse wieder auf einen Prozess abgebildet.

- Anders als beim Ansatz von Agrawal et al. verwenden die beiden im folgenden Abschnitt vorgestellten Ansätze von Hwang und Yang sowie Golani und Pinter Informationen über Zeitstempel (Totalordnung der Ereignisse des Logs) von Prozessen zur Aufdeckung von Parallelitäten. Auf diese Weise versucht auch der in dieser Arbeit in Kapitel 4 vorgestellte Ansatz auf Parallelitäten zu schließen.

3.1.2.2 Ansätze nach Hwang und Yang sowie Golani und Pinter

Da sich die Ansätze von Hwang und Yang sowie von Golani und Pinter sehr stark ähneln, beschränkt sich diese Arbeit darauf, den Ansatz von Hwang und Yang zu erklären und Unterschiede zum Ansatz von Golani und Pinter aufzuzeigen. Der Ansatz von Hwang und Yang [17] ist vom Ablauf her sehr ähnlich wie der Ansatz von Agrawal et al. Ein gerichteter Graph, welcher alle Anwendungsfälle des Prozesses des Logs modelliert, wird abgeleitet, indem zunächst für jede Instanz die Menge von Prozess-Paaren, die in einer direkten Nachfolgerbeziehung bezüglich dieser Instanz stehen, ermittelt wird. Dieser Schritt kommt einer Generierung von einzelnen Graphen für Instanz gleich. Die Menge der direkten Nachfolgerbeziehungen in

den Instanzen stellen potentielle Kanten im resultierenden Prozessgraphen dar. Der resultierende Graph enthält daher alle Kanten, die durch die Nachfolgerbeziehungen der Instanzen entstehen. Es werden anschließend alle Kanten zwischen Prozessen entfernt, deren Richtungen sich in mindestens zwei Instanzen überkreuzen. Der Ansatz von Golani und Pinter [18, 19] kann als eine Mischung der beiden oben vorgestellten Ansätze betrachtet werden. Das Vorgehen des Ansatzes entspricht im Wesentlichen dem Vorgehen von Hwang et al. Der wesentliche Unterschied besteht darin, dass Golani und Pinter zunächst nur die Ableitung von azyklischen Graphen vorsehen. Daher können sie sowohl zeitliche Überschneidungen von Prozessen als auch Verschränkung von Prozessen als Zeichen von Parallelität deuten, da eine Verschränkung von Prozessen zu einem Zyklus im resultierenden Modell führen würde. Nachdem ein Graph in ähnlicher Weise wie bei dem Ansatz von Hwang und Yang erzeugt wird, folgt bei dem Ansatz von Golani und Pinter daher ein Schritt, bei dem stark zusammenhängende Teile des Graphen entfernt werden. Ein wesentlicher Unterschied zwischen dem Ansatz von Agrawal et al. und dem von Golani und Pinter sowie Hwang und Yang ist, dass beim Ansatz von Agrawal et al. zunächst auch transitive Nachfolgerbeziehungen berücksichtigt werden. Für Details wird auf die Arbeiten von Hwang und Yang sowie Golani und Pinter verwiesen. Für den Umgang mit unvollständigen Daten verwenden alle drei vorgestellten Ansätze Schwellenwerte, um die Relevanz von Nachfolgerbeziehungen zu bewerten.

3.1.3 Kontrollfluss-Verfahren nach van der Aalst et al.

Auch die Verfahren von van der Aalst et al. versuchen den Kontrollfluss eines Prozesses aus den Logdaten zu rekonstruieren. Zur Modellierung des Prozessmodells werden größtenteils Petri-Netze verwendet. Van der Aalst et al. forschen seit über 15 Jahren an Process Mining Algorithmen der Kontrollflussperspektive. Aus den Forschungsaktivitäten sind zahlreiche Verfahren hervorgegangen, welche nach den folgenden Bewertungskriterien eingeordnet werden können.

3.1.3.1 Bewertungskriterien

Algorithmen, die ihren Fokus auf die Konstruktion des Kontrollflusses eines Prozesses richten, lassen sich vor allem in der Hinsicht bewerten, welche und wie viele verschiedene imperative Prozessmodellelemente sie in der Lage sind zu rekonstruieren und wie sie mit Fehlern und Unvollständigkeiten in Logdateien umgehen. [20] nennt grundlegende Kriterien, welche im folgenden Abschnitt kurz erläutert werden.

Robustheit

Nach [20] stellt das Umgehen mit unsauberen, d.h. verrauschten bzw. unvollständigen Informationen des Ereignis-Logs ein wichtiges Bewertungskriterium dar. Der Begriff Rauschen bezeichnet Störungen, wie beispielsweise fehlerhafte Zeitstempel oder fehlende Teile einer Ereignisfolge einer Prozessinstanz. Ursachen für Rauschen in Logdateien sind auftretende Ausnahmen im Prozessablauf sowie falsch protokollierte Ereignisse. Sind Verfahren in der Lage, trotz Vorhandensein von Rauschen die Logdateien zu verarbeiten und richtige Ergebnisse zu liefern, werden diese als robust bezeichnet, andernfalls als nicht robust.

Erkennbare Prozessmodellelemente

Prozess-Mining Ansätze werden weiterhin hinsichtlich ihrer Mining-Fähigkeiten betrachtet. Die wichtigsten imperativen Modellierungskonstrukte sind im Folgenden kurz aufgeführt.

Sequenzen

Das Erkennen von sequentiellem Verhalten stellt eine Minimalanforderung dar und wird praktisch von jedem Algorithmus beherrscht.

Paralleles Verhalten
Ähnlich wie bei Sequenzen stellt auch die Erkennung von parallelem Verhalten eine Grundanforderung an Process Mining Algorithmen dar.

Kontrollflussverzweigungen (engl. split bzw. join)
Auch die Detektion von Kontrollflussverzweigungen stellt laut [20] eine grundlegende Voraussetzung an einen Process-Mining Algorithmus dar.

Schleifen
Bei Schleifen handelt es sich um ein erstes, nicht einfach zu erkennendes Konstrukt. Algorithmen lassen hierbei dadurch unterscheiden, ob sie in der Lage sind, beliebig, gar keine, oder nur eingeschränkt Schleifen zu erkennen.

Mehrfach auftretende Aktivitäten
Ein weiteres Bewertungskriterium ist, ob die Ansätze in der Lage sind, mehrfach auftretende Aktivitäten korrekt zu erkennen. Mehrfach auftretende Aktivität bedeutet, dass mehrere Prozessschritte die gleiche Beschriftung besitzen. Das Problem hierbei ist, dass die meisten Mining-Verfahren diese mehrfach auftretenden Aktivitäten als eine einzelne auffassen.

Nicht protokollierte Ereignisse (engl. hidden activities)
Nicht protokollierte Ereignisse lassen sich entsprechend schwer von einem Algorithmus erkennen. Solche Ereignisse können Routing- oder Synchronisationereignisse (z.B. splits oder joins) sein, aber auch dazu führen, dass ein Prozessschritt übersprungen wird.

Non-free-Choice Konstrukte
Bei non-free-Choice Konstrukten hängt das Ergebnis einer Entscheidung davon ab, ob ein bestimmtes Ereignis vorausgegangen ist oder nicht. Man unterscheidet dabei, ob ein direkt vorangegangenes Ereignis (lokal non-free choice) oder ein nicht direkt vorangegangenes Ereignis das Ergebnis einer Entscheidung beeinflusst hat (non-lokal non-free choice construct).

3.1.3.2 Alpha-Algorithmus

Im folgenden Abschnitt wird der α-Algorithmus erläutert (vgl. [5]), welcher den Prozessablauf und die Reihenfolge der Aktivitäten innerhalb einer Prozessinstanz betrachtet. Der Ansatz lässt sich wie folgt klassifizieren: Das Verfahren strebt eine Entdeckung eines Prozessmodells an, wobei es die Kontrollfluss-Perspektive untersucht, in die Analyse mehrere oder alle Prozessinstanzen einbezieht, die Vergangenheitsdaten aus einer Logdatei betrachtet und als Ergebnis Informationen in Form von Entscheidungsregeln liefert.

Inputbeschreibung
Der Algorithmus benötigt Informationen über die Prozess- und Prozessinstanzzugehörigkeit der im Event-Log auftretenden Ereignisse. Es werden dabei nur Zusammenhänge zwischen Aktivitäten pro Prozessinstanz untersucht. Um eine korrekte Rekonstruktion des Prozessmodells gewährleisten zu können, müssen sämtliche Aktivitäten mindestens einmal in der Logdatei als Ereignis verzeichnet sein (Vollständigkeit). Des Weiteren erlaubt der Algorithmus keine Fehler und Ausnahmen (Freiheit von Rauschen). Eine Ereignis-Logdatei wird durch die Notation des α-Algorithmus wie folgt beschrieben: T beschreibt eine Menge von Aktivitäten, $\sigma \in T^*$ bezeichnet eine Ereigniskette (engl. event trace), d. h. eine Abfolge von Aktivitäten und $W \subseteq T^*$ ist eine Ereignis-Logdatei, d. h. eine Menge aus Ereignisketten. Hier

ist zu beachten, dass es sich um eine Menge und keine Multimenge handelt, folglich spielt die Anzahl des Auftretens einer Ereigniskette keine Rolle beim Ablauf des Algorithmus.

Funktionsweise

Angenommen die Auswertung eines Logs liefere die Einträge $W = \{ABCD, ACBD, AED\}$ als Menge an Ereignisketten. Im nächsten Schritt wird nach kausalen Zusammenhängen zwischen zwei Aktivitäten x und y gesucht. Die binären Beziehungen zwischen zwei Aktivitäten unterscheidet der Algorithmus wie folgt:

- Aktivitäten, die direkt aufeinander folgen (Notation $x >_W y$)
- Direkte kausale Beziehungen, d. h. stets folgt y auf x, aber nie x auf y (Notation: $x \rightarrow_W y$)
- Potentielle Parallelität wird angenommen wenn sowohl $x >_W y$ als auch $y >_W x$ gefunden wird (Notation: $x \parallel_W y$)
- Transitionspaare, welche nie direkt aufeinander folgen, keine kausale Beziehung zueinander besitzen und bei welchen eine Parallelität als unwahrscheinlich gilt (Notation: $x \#_W y$)

Anhand dieser Beziehungen überführt der Algorithmus die Informationen aus der Ereignis-Logdatei in ein Prozessmodell. Als Prozessmodellierungssprache verwenden [5] Petrinetze. Auf eine detaillierte Erläuterung der Darstellung von Prozessmodellen in Petrinetzen wird in dieser Arbeit aus Zeitgründen verzichtet. Eine ausführliche Beschreibung dieser Abbildung wird jedoch in [21] gegeben. Betrachtet man die obige Event-Logdatei W und sucht dort sämtliche Beziehungen zwischen Aktivitäten, erhält man beispielsweise $A >_W B$, $A >_W C$, $A >_W E$, $B >_W C$, $C >_W B$. Diese Aktivitäten folgen in der Logdatei mindestens einmal direkt aufeinander. Man erhält außerdem anhand der obigen Regeln $A \rightarrow_W B$, $A \rightarrow_W C$, $A \rightarrow_W E$ sowie $B \parallel_W C$. Eine Übersicht der Abbildungsregeln wird in Abbildung 2 gegeben.

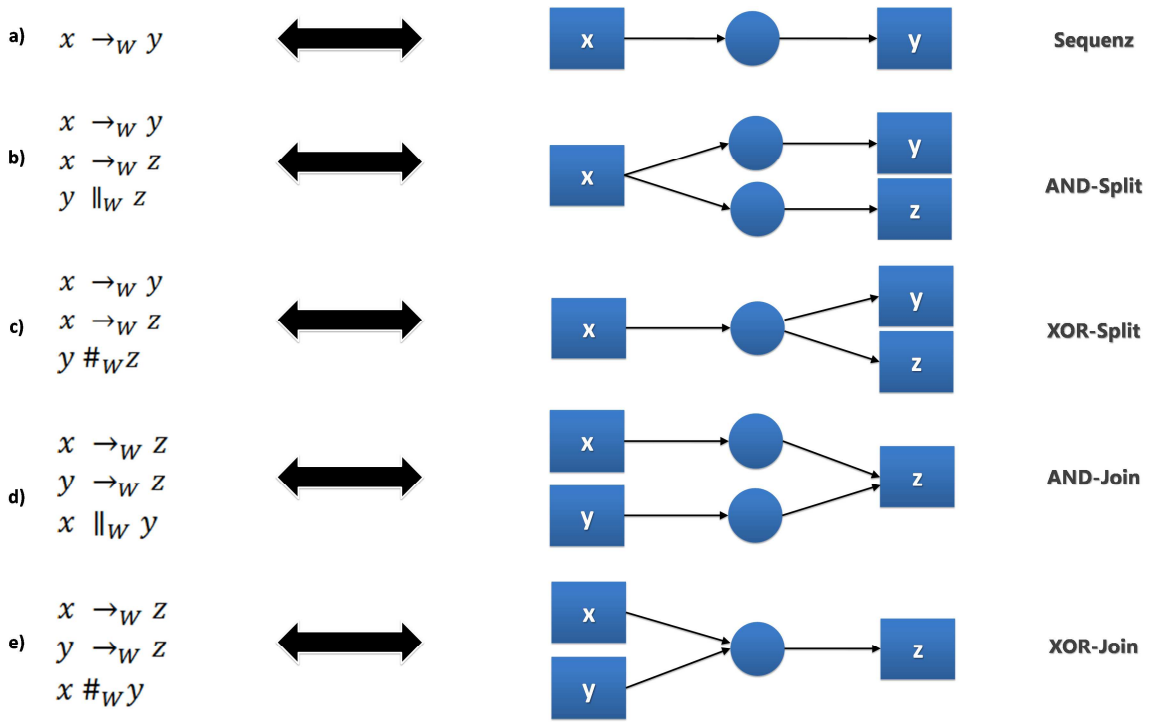

Abbildung 2: Abbildungsregeln des α-Algorithmus (vgl. [5])

Falls alle auf der linken Seite genannten Beziehungen zwischen den einzelnen Aktivitäten zutreffen, bildet der Algorithmus das angegebene Petrinetz-Konstrukt. Betrachtet man beispielsweise Fall b) der Abbildung 2: Aus der Analyse einer Logdatei ging hervor, dass Aktivität y stets auf x folgt ($x \longrightarrow_W y$), dass Aktivität z stets auf x folgt ($x \longrightarrow_W z$) und dass eine potentielle Parallelität zwischen y und z besteht ($y \parallel_W z$). Der Algorithmus konstruiert das korrespondierende Petrinetz, welches die Parallelität zwischen y und z zum Ausdruck bringt. Ein vollständiges Prozessmodell, welches durch den Algorithmus konstruiert wurde zeigt Abbildung 3. Hier zeigt sich bereits ein erster Nachteil des α-Algorithmus, denn er ist nicht in der Lage sog. nicht-protokollierte Ereignisse zu entdecken. Das vollständig-korrekte Prozessmodell enthielte ein AND-split und ein AND-join Ereignis, um die korrekte Semantik des Ablaufs darstellen zu können. Da der α-Algorithmus den Prozess als Petrinetz abbildet, ist eine Darstellung dieser nicht-protokollierten Ereignisse nicht möglich und nur implizit gegeben. Abbildung 3 zeigt in der rechten Spalte ansatzweise welcher Semantik das jeweilige Petrinetz Konstrukt entspricht.

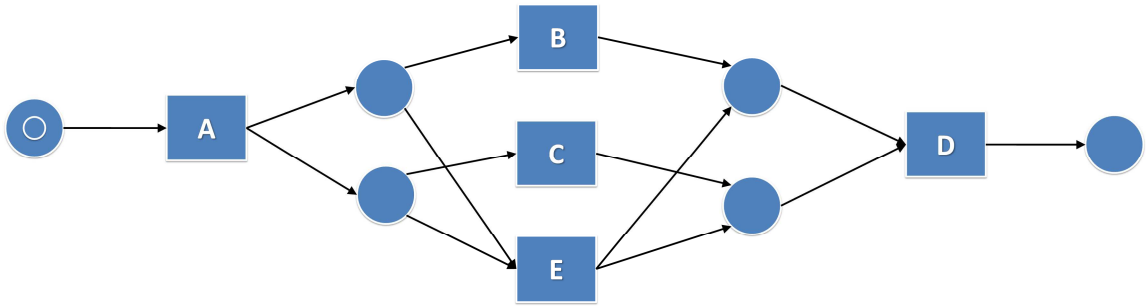

Abbildung 3: Konstruiertes Prozessmodell des α-Algorithmus (vgl. [5])

Ein weiteres Problem des Ansatzes ist, dass keine Berücksichtigung der Häufigkeiten der Ereignisse stattfindet. Diese Tatsache ist darauf zurückzuführen, dass es sich bei der Logdatei W nur um eine einfache Menge und keine Multimenge von Ereignisketten handelt. Der Algorithmus ist daher sehr anfällig für Rauschen, d. h. er ist nicht in der Lage aus fehlerhaften Logdateien ein korrektes Prozessmodell zu generieren. Aufgrund der Tatsache, dass lediglich binäre Beziehungen betrachtet werden, kann der Algorithmus über mehr als zwei Aktivitäten spannende Abhängigkeiten nicht korrekt erfassen. Diese Problematik zeigt sich darin, dass vor allem kurze Schleifen und nicht lokale Non-free-Choice-Konstrukte nicht korrekt rekonstruiert werden können. Tabelle 2 fasst die Fähigkeiten und Grenzen des α-Algorithmus zusammen. Weiterentwicklungen wie der α+ und der α++ Algorithmus bieten zu einigen Problemen des α-Ansatzes Lösungsansätze. Der α+ Ansatz (vgl. [22]) ist in der Lage, beliebige Schleifen zu rekonstruieren und behebt das Problem der Erkennung von kurzen Schleifen. Der Fokus des α++ Ansatzes (vgl. [23]) liegt auf der Erkennung von Non-free-choice Konstrukten was dadurch erreicht wird, dass größere Beobachtungsfenster verwendet werden und die Betrachtung der Abhängigkeiten der Aktivitäten nicht nur auf binäre Beziehungen beschränkt ist.

Robustheit	Gering
Erkennbare Prozessmodellelemente	
- Sequenzen	Ja
- Paralleles Verhalten	Ja
- Kontrollflussverzweigungen	Ja
- Schleifen	keine kurzen Schleifen, beliebig bei α+ Algorithmus
- Mehrfach auftretende Aktivitäten	Nein
- Nicht protokollierte Ereignisse	Nein

- Non-free-Choice-Konstrukte	Lokal (beliebig bei α++ Algorithmus)

Tabelle 2: Fähigkeiten und Grenzen des α-Algorithmus (in Anlehnung an [20])

3.1.3.3 Heuristics-Miner Algorithmus

Der Heuristic-Miner Algorithmus (vgl. [24]) kann als eine Erweiterung des α-Algorithmus aufgefasst werden, da er die gleichen Nachfolgebeziehungen verwendet. Zum Erkennen von kausalen Abhängigkeiten und parallelem Verhalten zwischen zwei Ereignissen werden allerdings die Häufigkeiten der Nachfolgerrelation in der Logdatei berücksichtigt.

Inputbeschreibung

Der vorliegende Ansatz erfordert genau wie der α-Algorithmus Informationen über die Aktivitäts- und Prozessinstanzzugehörigkeit der im Event-Log auftretenden Ereignisse. Des Weiteren muss auch jeder zu erkennende Prozessschritt mindestens einmal im Ereignislog vorkommen (Vollständigkeit). Im Gegensatz zum α-Ansatz wird bei dem heuristischen Ansatz nicht angenommen, dass die Informationen aus der Ereignis-Logdatei frei von Rauschen sind. Der Algorithmus kann also sehr gut mit Fehlern und Ausnahmeerscheinung in der Logdatei umgehen. Die Abbildung der Logdatei erfolgt bei diesem Ansatz auf eine Multimenge W.

Funktionsweise

Der Heuristic-Miner Algorithmus berücksichtigt neben der rein binären Information, ob eine Nachfolgebeziehung besteht oder nicht, ebenfalls die Häufigkeiten der auftretenden Ereignisse. Diese häufigkeitsbasierte Metrik wird verwendet, um die Verlässlichkeit zu bestimmen, mit der eine Abhängigkeitsbeziehung zwischen zwei Aktivitäten a und b besteht (Notation: $a \Rightarrow_W b$). Sämtliche andere Beziehungen sind identisch mit denen des α-Algorithmus. Die eingeführte Metrik ist definiert als:

$$a \Rightarrow_W b = \left(\frac{|a >_W b| - |b >_W a|}{|a >_W b| + |b >_W a| + 1} \right) \qquad (3.1)$$

$|a >_W b|$ ist dabei die Auftrittshäufigkeit der direkten Nachfolgerbeziehung $a >_W b$ in der Logdatei W. Der Wert dieser Verlässlichkeit liegt stets zwischen -1 und 1. Da bei Schleifen der Länge 1 und 2 der Wert für die Verlässlichkeit sehr klein werden würde, geben [24] für Schleifen dieser Längen zusätzliche Metriken an:

$$a \Rightarrow_W a = \left(\frac{|a >_W a|}{|a >_W a| + 1} \right) \qquad (3.2)$$

$$a \Rightarrow_{2W} b = \left(\frac{|a >>_W b| - |b >>_W a|}{|a >>_W b| + |b >>_W a| + 1} \right) \qquad (3.3)$$

Die Beziehung $a \gg_W b$ liegt vor, wenn im Log W bei einer Prozessinstanz die Teilfolge aba auftritt (Schleife der Länge 2). Der Algorithmus benutzt obige Metriken und geht wie folgt vor: Im ersten Schritt wird ein Abhängigkeitsgraph (engl. dependency graph) berechnet. Dieser gibt die Abhängigkeiten zwischen einzelnen Aktivitäten an. Grundlage der Berechnung des Abhängigkeitsgraphen sind die obigen Formeln bzw. Metriken (3.1), (3.2) und (3.3). Anhand der errechneten Werte und drei Schwellenwerten entscheidet der Algorithmus, ob es sich bei einer Beziehung zwischen zwei Aktivitäten um eine reale kausale Abhängigkeit, um Rauschen oder um eine Ausnahme handelt. Es liegt genau dann eine kausale

Abhängigkeit vor, wenn der errechnete Wert aus (3.1) bzw. (3.2) und (3.3) über dem sog. Abhängigkeitsschwellenwert (engl. dependency threshold) liegt, die Auftrittshäufigkeit der Beziehung größer als der Schwellenwert für korrekte Beobachtungen (engl. positive observation threshold) ist und der Unterschied der berechneten Abhängigkeit zur besten bisher berechneten Abhängigkeit geringer als der RTB-Schwellenwert (engl. relative-to-best threshold) ist. Als Beispiel wird die Logdatei $W = \{ABCD^9, ACBD^9, AED^9, ABCED, AD, AECBD\}$ betrachtet (vgl. [24]). Die Logdatei enthält in diesem Beispiel dreißig Ereignisketten (neun für jede der drei „richtigen" Pfade und drei inkorrekte Ketten). Berechnet man sämtliche Werte der Metriken, ergibt sich folgende Wertematrix:

$\Rightarrow_W$	A	B	C	D	E
A	0.0	0.909	0.900	0.500	0.909
B	0.0	0.0	0.0	0.909	0.0
C	0.0	0.0	0.0	0.900	0.0
D	-0.500	-0.909	-0.909	0.0	-0.909
E	0.0	0.0	0.0	0.909	0.0

Tabelle 3: Beispiel einer Wertematrix des Heuristic-Miner Algorithmus (vgl. [24])

Anhand der Werte der Matrix wird ersichtlich, dass es sich beispielsweise bei der Beziehung AD um einen Fehler handeln könnte (Wert 0.5). Der sich ergebende Abhängigkeitsgraph ist in Abbildung 4 dargestellt. Den einzelnen Aktivitäten sind in Klammern deren Auftrittshäufigkeiten beigefügt. Die Beziehung AD findet keine Berücksichtigung im Graphen, da der Abhängigkeitsschwellenwert in diesem Beispiel größer als 0.5 gewählt wurde.

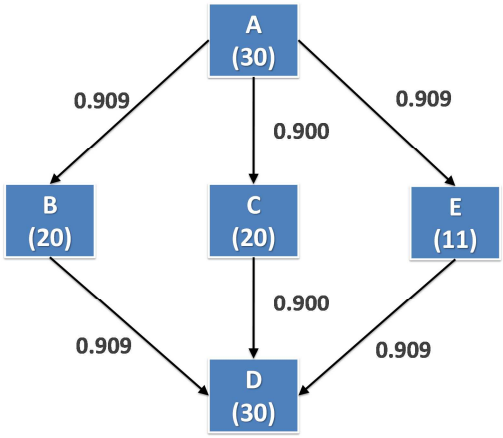

Abbildung 4: Abhängigkeitsgraph des Heuristic-Miner Algorithmus (vgl. [24])

Die Problematik der nicht-protokollierten Ereignisse umgeht der Ansatz dadurch, dass die explizite Darstellung der Konstrukte vermieden wird. Das Prozessmodell wird anstatt in Form eines Petrinetzes anhand von sog. Kausalitätsmatrizen repräsentiert. Tabelle 4 zeigt die Kausalitätsmatrix zum laufenden Beispiel.

Aktivität	Input	Output
A	/	$(B \vee E) \wedge (C \vee E)$
B	A	D
C	A	D
D	$(B \vee E) \wedge (C \vee E)$	/
E	A	D

Tabelle 4: Kausalitätsmatrix des Heuristic-Miner Algorithmus (vgl. [24])

Die Kausalitätsmatrix gibt für jede Aktivität die Input- und Output-Aktivitäten an. Aktivität A hat als Startknoten beispielsweise keinen Input. Nach Ausführung von A wird dessen Output $(B \vee E) \wedge (C \vee E)$ aktiviert. Anhand einer weiteren Metrik (3.4) kann erschlossen werden, um welche Art von Beziehung und somit um welche Art von split- und join-Konnektor es sich handelt.

$$a \Rightarrow_W b \wedge c = \left(\frac{|b >_W c| + |c >_W b|}{|a >_W b| + |a >_W c| + 1} \right) \qquad (3.4)$$

Berechnet man beispielsweise für die Aktivitäten A, B und C den Wert der Formel (3.4) ergibt sich $A \Rightarrow_W B \wedge C = \big((10 + 10) \,/\, (10 + 9 + 1)\big) = 1.0$. Ein Wert nahe bei 1.0 weist darauf hin, dass sich B und C in einer AND-Relation befinden. Ist der errechnete Wert nahe bei 0, wird eine XOR-Relation angenommen. Anhand der errechneten Ergebnisse kann das implizit gegebene Prozessmodell bei Bedarf in ein Petrinetz übersetzt werden (vgl. [24]). Tabelle 5 gibt einen abschließenden Überblick über die Fähigkeiten des Heuristic-Miner Algorithmus.

Robustheit	Hoch
Erkennbare Prozessmodellelemente	
- Sequenzen	Ja
- Paralleles Verhalten	Ja
- Kontrollflussverzweigungen	Ja
- Schleifen	beliebig
- Mehrfach auftretende Aktivitäten	Nein
- Nicht protokollierte Ereignisse	Überspringen von Schritten
- Non-free-Choice-Konstrukte	Lokal

Tabelle 5: Zusammenfassung des Heuristic-Miner Algorithmus (in Anlehnung an [20])

3.1.3.4 Genetisches Process Mining

Im folgenden Abschnitt wird als letztes Verfahren von van der Aalst et al. der genetische Process Mining Algorithmus (vgl. [25]) betrachtet. Die Zielsetzung des genetischen Algorithmus ist die Behandlung problematischer Kontrollflusskonstrukte, welche den vorher genannten Verfahren Probleme bereitet haben.

Inputbeschreibung
Das genetische Process Mining erfordert keinerlei besondere Voraussetzungen bzw. trifft keine einschränkenden Grundannahmen an die vorliegende Logdatei und ist damit als besonders robust einzustufen.

Funktionsweise
Der genetische Ansatz versucht nicht lokale binäre Beziehungen im Ereignislog zu entdecken und zu optimieren, sondern betrachtet den zugrundeliegenden Prozess global. Ziel ist die Evolution von kompletten vollständigen Prozessmodellen. Der Algorithmus funktioniert dabei grundlegend nach folgendem Ablauf: Das Verfahren beginnt mit einer initialen Population an Prozessmodellen (sog. Individuen). Diese Modelle entsprechen jeweils einer Kausalmatrix und werden beispielsweise unter Verwendung des im vorherigen Abschnitt beschriebenen Heuristic-Miner Algorithmus erzeugt. Anschließend wird für jedes Individuum einer Generation eine Fitnesszahl berechnet welche angibt, wie gut ein Prozessmodell das in der Logdatei spezifizierte Verhalten wiedergeben kann. Hierbei werden die Verhältnisse der fehlerfrei geparsten Prozessschritte und der nicht fehlerfrei geparsten Prozessschritte zur Gesamtzahl der in der Logdatei vorhandenen Aktivitäten berücksichtigt. Die detaillierte Beschreibung der Berechnung der Fitnesszahl F eines Modells findet sich in [25]. Die sog. Population entwickelt sich anschließend weiter, indem die besten Modelle einer Generation in die Folgegeneration übernommen werden (Selektion) und neue Individuen auf Basis der selektierten Modelle erzeugt werden. Hierbei erfolgt die Anwendung genetischer Operatoren, zum einen Kreuzung, d.h. Kombination von Teilen mehrerer Modelle, zum anderen Mutation, d. h. zufällige Veränderung von Modellen. [25] gibt eine detaillierte algorithmische Beschreibung der beiden genetischen Operatoren an. Im Rahmen dieser Arbeit werden die Operatoren im Folgenden kurz beschrieben.

Kreuzung
Kreuzung ist ein genetischer Operator der existierende Teile von Modellen der aktuellen Population miteinander kombiniert. Der Operator bezieht alle Individuen einer Population in den Suchraum mit ein. Kreuzung kann folgende Auswirkungen auf ein Prozessmodell-Individuum haben: Verlieren von Einzelschritten, Hinzufügen von Schritten, Verändern von Abhängigkeitsbeziehungen zwischen Aktivitäten oder Hinzufügen von Beziehungen. Es wird dabei zufällig ein Prozessschritt als Kreuzungspunkt aus den beiden miteinander gekreuzten Elternmodellen ausgewählt.

Mutation
Der Mutationsoperator zielt darauf ab neue Elemente in der aktuellen Population einzufügen. Der Operator führt dabei eine der folgenden Aktionen auf einem Prozessmodell-Individuum aus: (i) Zufälliges Hinzufügen einer Aktivität zu Ein- oder Ausgangsmengen eines Schrittes, (ii) Zufälliges Löschen einer Aktivität oder (iii) zufälliges Neuaufteilen der Elemente in den Ein- und Ausgangsmengen.

Der Algorithmus terminiert, wenn ein Prozessmodell die Fitnesszahl F = 1 erlangt, wenn eine vorgegebene Anzahl an Generationen erzeugt wurde oder wenn sich das beste Individuum innerhalb eines längeren Zeitraums nicht mehr verändert hat. Eine zusammenfassende Darstellung der Fähigkeiten des Verfahrens gibt Tabelle 6.

Robustheit	hoch
Erkennbare Prozessmodellelemente	
- Sequenzen	Ja
- Paralleles Verhalten	Ja
- Kontrollflussverzweigungen	Ja

- Schleifen	beliebig
- Mehrfach auftretende Aktivitäten	Aktivitäten
- Nicht protokollierte Ereignisse	beliebig
- Non-free-Choice-Konstrukte	beliebig

Tabelle 6: Zusammenfassung des Genetischen Process Mining (in Anlehnung an [20])

Es ist festzuhalten, dass der Algorithmus zwar sehr gute Ergebnisse liefert und in der Lage ist nahezu jedes Prozesselement korrekt zu erkennen, jedoch aufgrund der komplexen und umfangreichen genetischen Operationen teilweise sehr lange Laufzeiten aufweist. Dies geht aus den Benchmark-Ergebnissen aus [25] hervor.

3.1.3.5 Zusammenfassung und Vergleich

Abschließend wird ein kurzer Überblick über die behandelten Algorithmen der Kontrollflussperspektive gegeben. Der α-Algorithmus kann als Pionier der Process Mining Verfahren betrachtet werden. Er ist in der Lage grundlegende imperative Prozessmodellelemente wie Sequenzen, paralleles Verhalten, Verzweigungen und Schleifen korrekt zu rekonstruieren. Die Weiterentwicklungen des α-Algorithmus (α+ und α++) ermöglichen außerdem eine Verbesserung der Erkennung von problematischen Konstrukten wie kurzen Schleifen und Non-free-Choice-Konstrukten. Nachteil der α-Ansätze ist die geringe Robustheit und somit die Anfälligkeit für fehlerhafte Loginformationen, was die Ansätze im realen Einsatz im Allgemeinen vor große Probleme stellt, da Fehler und Ausnahmen im täglichen Betrieb eines Informationssystems häufig auftreten. Der Heuristic-Miner Algorithmus versucht vor allem dieses Problem aufzugreifen und zu beheben, indem er eine häufigkeitsbasierte Betrachtung der entdeckten Beziehungen anstrebt und so robust gegenüber Ausnahmen und Fehlern in der Logdatei ist. Das Prozessmodell wird jedoch nicht explizit dargestellt, sondern ist nur in Form eines Abhängigkeitsgraphen in Kombination mit einer Kausalitätsmatrix gegeben. Für eine anschauliche Visualisierung des erzeugten Prozessmodells sind hier also weitere Verarbeitungsschritte notwendig. Zuletzt wurde der genetische Process Mining Algorithmus betrachtet, welcher nicht versucht lokale Beziehungen zwischen Aktivitäten zu ermitteln, sondern versucht durch eine globale Betrachtung ein passendes Prozessmodell zu konstruieren. Das Verfahren zeichnet eine hohe Robustheit gegenüber Ausnahmen und Fehlern in der Logdatei, sowie eine große Vielfalt der erkennbaren Prozessmodellelemente aus. Hier liefert der genetische Ansatz eindeutig die besten Ergebnisse. Der Algorithmus benötigt im Allgemeinen jedoch eine wesentlich längere Laufzeit als die zuvor genannten Algorithmen. Da es sich bei den α-Ansätzen um vollständig determinierte Algorithmen handelt, liegt deren Erkennungsrate bei 100 %. Das bedeutet, dass wenn die Voraussetzungen der Algorithmen erfüllt sind, auf jeden Fall das richtige Prozessmodell mit den erkennbaren Prozesselementen rekonstruiert wird. Die Erkennungsraten des Heuristic-Miners und des genetischen Process Mining hängen stark von den gewählten Parametern, wie den Werten der Schwellenwerte und der maximalen Generationsanzahl, ab. Gute Ergebnisse sind beim genetischen Ansatz häufig nur bei einer hohen Generationsanzahl zu erwarten. Welcher Ansatz für ein vorliegendes Problem zu wählen ist, hängt somit stark von den gegebenen Voraussetzungen und den gewünschten Ergebnissen ab. Die ermittelten Fähigkeiten und Grenzen der betrachteten Process Mining Verfahren sind in Tabelle 7 abschließend dargestellt.

Algorithmus	α-Algorithmus	α+ Algo.	α++ Algo.	Heuristic M.	Genet. PM
Robustheit	gering	gering	gering	hoch	Hoch
Erkennbare Prozessmodellelemente					
- Sequenzen	Ja	Ja	Ja	Ja	Ja
- Paralleles Verhalten	Ja	Ja	Ja	Ja	Ja
- Kontrollflussverzweigungen	Ja	Ja	Ja	Ja	Ja
- Schleifen	Keine kurzen	beliebig	beliebig	beliebig	beliebig
- Mehrfach auftretende Aktivitäten	nein	nein	nein	nein	Aktivitäten
- Nicht protokollierte Ereignisse	nein	nein	nein	Überspringen	beliebig
- Non-free-Choice-Konstrukte	lokal	lokal	beliebig	lokal	beliebig
Laufzeit	gering	gering	gering	gering	hoch
Erkennungsrate	100 %	100 %	100 %	abhängig von gewählten Einstellungen	abhängig von gewählten Einstellungen

Tabelle 7: Überblick über verschiedene Algorithmen (in Anlehnung an [20])

3.1.4 Betrachtung der organisatorischen Perspektive

Exemplarisch für ein Verfahren welches isoliert die organisatorische Perspektive eines Prozesses betrachtet wird der Social-Network Miner (vgl. [26]) erläutert. Ereignis-Logdateien speichern typischerweise Informationen über die ausführende oder verantwortliche Person einer Aktivität. Die Analyse der gespeicherten Information ermöglicht es Beziehungen zwischen den beteiligten Personen zu identifizieren. Ziel ist es ein Soziogramm zu erzeugen, das als Eingabe für Tools im Bereich der SNA (Social Network Analysis) dienen kann.

3.1.4.1 Social Network Analysis

Es gibt zahlreiche Anwendungsbereiche von SNA Tools. Allen Vorgehensweisen gemeinsam ist jedoch der Startpunkt der Analyse. Als Eingabe dient ein Graph, in dem Knoten Personen und Kanten Beziehungen repräsentieren. Dieser Graph heißt Soziogramm. Im mathematischen Sinne ist ein Soziogramm ein Graph (P, R) mit P als Menge von Individuen und $R \subseteq PxP$. Wenn der Graph zusätzlich gewichtet ist, weißt eine zusätzliche Funktion jedem Element in R einen Wert zu. Dieser Wert gibt an wie stark eine Beziehung ist. SNA definiert mehrere Eigenschaften, die in einem Soziogramm enthalten sein können. Betrachtet man den Graphen als Ganzes, kann beispielsweise die Dichte (engl. density) betrachtet werden. Dabei handelt es sich um die Anzahl an Elementen in R geteilt durch die maximale Anzahl an Elementen. Betrachtet man ein spezielles Individuum des Graphen, kann beispielsweise die Zentralität (engl. centrality) der Person analysiert werden. Dieses Maß gibt an, wie zentral eine Person im Graphen gelegen ist, d.h. wie weit der Weg zu den anderen Individuen ist. Weitere Metriken werden in [26] erläutert.

3.1.4.2 Der Social Network Miner

Die Aufgabe des Process Mining im Bereich der organisationsorientierten Perspektive ist es nun, aussagekräftige Soziogramme aus Ereignis-Logdateien zu extrahieren, welche den SNA Tools als Input-Information dienen können. [26] definieren hierfür vier verschiedene Metriken um Beziehungen zwischen Individuen darzustellen: (1) Metriken auf Basis von Kausalität, (2) Metriken auf Basis gemeinsamer Instanzen, (3) Metriken auf Basis gemeinsamer Aktvitäten und (4) Metriken auf Basis spezieller

Ereignistypen. Metriken auf Basis von Kausalität betrachten für einzelne Fälle wie Arbeit von Person zu Person übergeben wird. Ein Beispiel solch einer Metrik ist „handover of work“. Innerhalb einer Prozessinstanz findet eine Übergabe von Arbeit zwischen Person i und Person j genau dann statt, wenn es zwei aufeinanderfolgende Aktivitäten gibt, wobei die erste Aktivität von i und die zweite Aktivität von j ausgeführt wurde. Die Metrik „handover of work“ liefert beispielsweise ein in Abbildung 5 dargestelltes Soziogramm, welches Beziehungen zwischen Mitarbeitern darstellt.

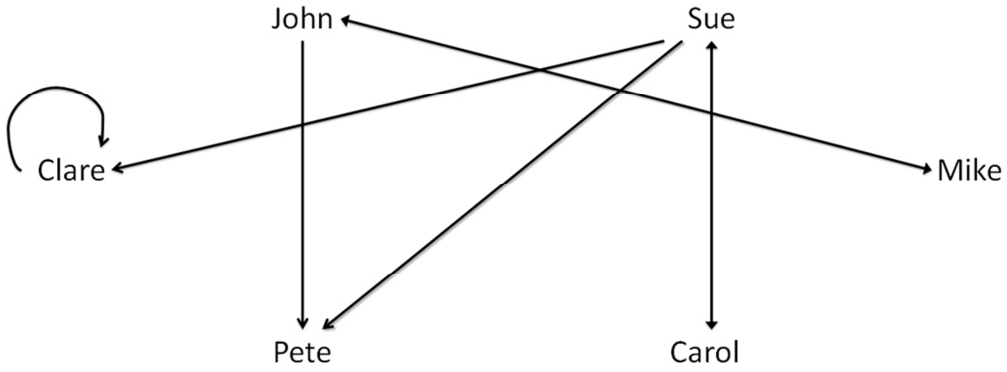

Abbildung 5: Beispiel eines Soziogramms "handover of work" (vgl. [26])

Es wird ersichtlich, dass bei Carol und Sue eine gegenseitige Übergabe von Arbeit stattfindet, wohingegen John nur Arbeit an Pete übergibt, nicht jedoch Pete an John. Metriken auf Basis gemeinsamer Instanzen ignorieren kausale Abhängigkeiten und zählen nur, wie oft zwei Individuen Aktivitäten derselben Prozessinstanz ausführen. Personen, die häufiger zusammen an Fällen arbeiten, haben höchstwahrscheinlich eine stärkere Beziehung zueinander als Personen, die nur selten an einem gleichen Fall arbeiten. Metriken auf Basis gemeinsamer Aktivitäten berücksichtigen inwieweit Individuen die gleichen Aktivitäten ausführen. Auch hier ist die Vermutung, dass Personen, die oft eine gleiche Tätigkeit ausführen, eine stärkere Beziehung zueinander besitzen. Metriken auf Basis spezieller Ereignistypen betrachten den Typ eines Ereignisses. Wenn beispielsweise Person i häufig nur Arbeit an Person j delegiert ist es realistisch, dass i in einer hierarchischen Beziehung zu j steht. Der spezielle Ereignistyp wäre in diesem Fall die Delegation von Arbeit. Eine quantitative Betrachtung der Metriken kann innerhalb dieser Arbeit aus Zeitgründen nicht erfolgen. Die einzelnen Definitionen der verschiedenen Metriken sind jedoch ausführlich in [26] aufgeführt.

3.1.5 Verfahren der datenorientierten Perspektive

Algorithmen der Kontrollflussperspektive liefern als Ergebnis Prozessmodelle, welche u.a. XOR-Verknüpfungen (somit Entscheidungspunkte) zwischen Prozessen darstellen. Es fehlt jedoch die Information, aufgrund welcher Gegebenheiten ein bestimmter Pfad an einem Entscheidungspunkt gewählt wird. Algorithmen der Datenperspektive analysieren beispielsweise, wie die Ausprägung von Datenattributen die Entscheidungspfade innerhalb des Gesamtprozesses beeinflussen. Als Beispielverfahren dieser Perspektive wird in dieser Arbeit das Decision Mining (vgl. [27]), auch als *decision point analysis* bezeichnet, betrachtet. Als Input für das Verfahren dienen Prozessmodelle, welche durch Verfahren zur Rekonstruktion des Kontrollflusses erzeugt wurden. Des Weiteren werden zusätzliche Informationen über die verarbeiteten Daten benötigt, welche im Ereignis-Log gespeichert sind. Der Algorithmus des Decision Mining beinhaltet zwei Schritte. Im ersten Schritt werden die Entscheidungspunkte innerhalb des gegebenen Prozessmodells identifiziert. Im zweiten Schritt wird untersucht, welche Datenwerte der verwendeten Attribute dazu führen, dass ein bestimmter Pfad eingeschlagen wird.

- Decision Mining stellt ein Process Mining Verfahren dar, welches zwei Perspektiven kombiniert betrachtet: Die Kontrollfluss- und die datenorientierte Perspektive. Es wird extrahiert wie sich der Kontrollfluss in Abhängigkeit von den verwendeten Daten verhält.

3.1.5.1 Auffinden von Entscheidungspunkten

Die Identifizierung von Entscheidungspunkten im Prozessmodell ist der grundlegende Schritt. Liegt das Modell als Petrinetz (Output des α-Algorithmus) vor, so ist ein Entscheidungspunkt dadurch gekennzeichnet, dass von einer Stelle mehrere Pfeile zu Transitionen ausgehen. Das Token kann nur einen der weiterführenden Pfade aktivieren. Ist das Modell hingegen in Form einer Kausalitätsmatrix gegeben (Output des Heuristic-Miners), so liegt genau dann ein Entscheidungspunkt nach einer Aktivität vor, wenn die Mächtigkeit deren Ausgabefunktion O größer als eins ist.

3.1.5.2 Ableiten von Entscheidungsregeln

Sind die Entscheidungspunkte ermittelt, werden Entscheidungsregeln für die jeweiligen Auswahlmöglichkeiten konstruiert. Dazu wird für jeden Entscheidungspunkt ein Klassifikationsproblem aufgestellt, wobei jeder Alternative eine Klasse zugeteilt wird. Anschließend wird die Ereignis-Logdatei untersucht und für jede Prozessinstanz analysiert, welcher Pfad gewählt wurde. Dazu wird die erste Aktivität nach einem Entscheidungspunkt betrachtet. Diese gibt Auskunft darüber, welcher Pfad in der jeweiligen Instanz tatsächlich gewählt wurde. Die Daten bis zum betrachteten Entscheidungspunkt werden der Klasse zugeordnet, welche vom Algorithmus entdeckt wurde. Es ergibt sich für jede Entscheidungsmöglichkeit eine Menge von Datenattribut-Ausprägungen, welche bei Prozessinstanzen entdeckt wurden, die einen bestimmten Pfad im Prozessmodell eingeschlagen haben. Das Klassifikationsproblem kann durch verschiedene Algorithmen gelöst werden. In [27] werden Entscheidungsbäume eingesetzt, um die Entscheidungsregeln zu visualisieren. Die Ergebnisse des Decision Mining Algorithmus können anschließend benutzt werden, um das bereits vorhandene Prozessmodell mit Entscheidungsregeln zu erweitern.

Somit wurde ein Einblick in bereits bestehende Process Mining Verfahren gegeben, welche sich auf die Rekonstruktion von imperativen Prozessmodellen spezialisiert haben oder die organisatorische Perspektive separat betrachten.

3.2 Semantikdefinitionen in der Prozessausführung

In Kapitel 2 wurden die unterschiedlichen Phasen des Business Process Lifecycle beleuchtet. Aufgrund der vielen unterschiedlichen Bereiche des BPM Lifecycle, sind zahlreiche verschiedene Personen an der Entwicklung und Implementierung von Prozessen beteiligt (Modellierer, Administratoren, Programmierer usw.). Begonnen wird im Allgemeinen mit der Modellierung von Prozessen. Die Anzahl an Modellierungselementen welche Modellierer verwenden können ist nicht fixiert und lässt Ihnen großen Spielraum. Meta-Modellierungsframeworks ermöglichen die Erweiterung und Anpassung von Prozessmodellierungssprachen durch domänenspezifische Konstrukte (vgl. [28]). Die Problematik besteht nun darin, die Intention, d.h. die Bedeutung, eines Modellierungselements allen beteiligten Personen klar zu machen. Es muss eine Eindeutigkeit der Bedeutungsdefinition gegeben sein, um die spätere Implementierung des Prozesses zu vereinfachen und Missverständnisse zwischen Modellierern und Programmierern auszuschließen.
Die Lösung dieses Problems wird durch Semantikdefinitionen erreicht. Die Dissertation von Christoph Günther [28] beschreibt detailliert den Einsatz von expliziten Semantikdefinitionen im Bereich der

Prozessausführung und vor allem deren Anwendung zur Validierung von Prozessen. Semantikdefinitionen ordnen Modellierungskonstrukten aus Prozessmodellen Bedeutungen zu, welche für sämtliche Bereiche des BPM gültig sind. Anhand dieser Definitionen entscheiden Modellierer welches Modellierungskonstrukt für ein bestimmtes Problem zu wählen ist und Programmierer wie das entsprechende Element zu implementieren ist. Dies ist zusammenfassend in Abbildung 6 dargestellt.

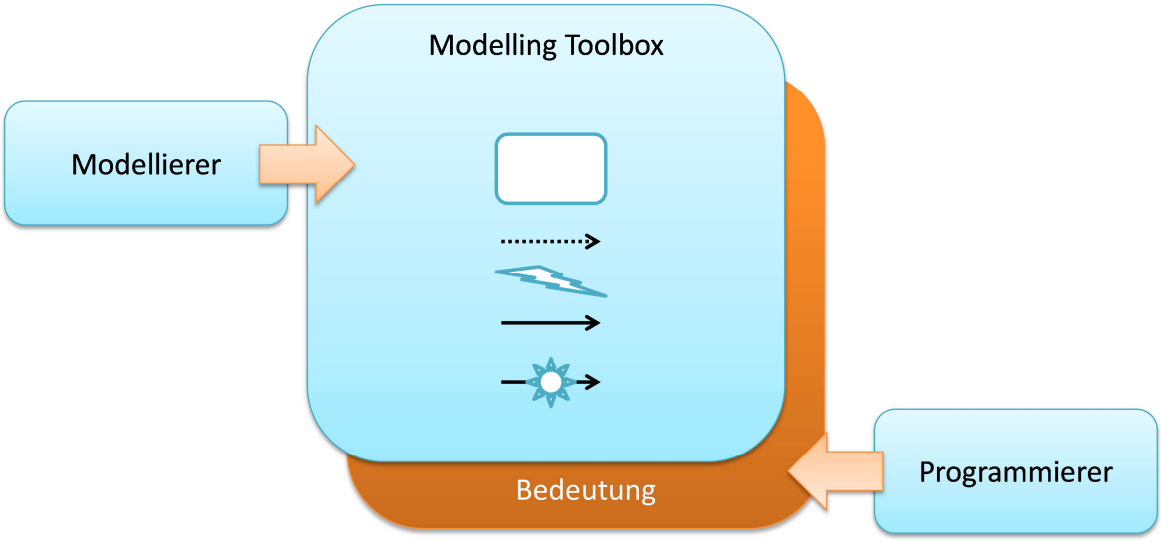

Abbildung 6: Semantikdefinitionen liefern die Bedeutung von Modellierungselementen [28]

[28] verwendet Semantikdefinitionen vor allem zur Validierung von ausgeführten Prozessen, d.h. es wird überprüft, ob der tatsächlich ausgeführte Prozess auch dem ursprünglich vom Modellierer beabsichtigten Prozess entspricht. Hierfür werden Logs ausgeführter Prozesse mit Hilfe von logischen Reasonern auf deren Gültigkeit überprüft. Das beschriebene Szenario wird in Abbildung 7 illustriert.

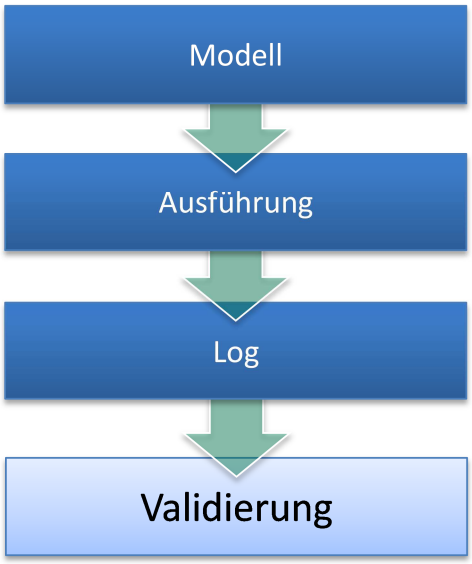

Abbildung 7: Validierung von ausgeführten Prozessen [28]

In dieser Arbeit wird in Kapitel 4 nun eine weitere Anwendungsmöglichkeit von Semantikdefinitionen im Bereich der Prozessausführung vorgestellt.

4 Process Mining auf Basis expliziter Semantikdefinitionen

In diesem Abschnitt wird ein neuer Process Mining Ansatz vorgestellt, welcher auf expliziten Semantikdefinitionen beruht. Dieser Ansatz wird im Folgenden als *SemanticPM* bezeichnet. Anstatt Semantikdefinitionen zur Validierung von Prozessen zu verwenden, werden bei diesem Verfahren aufgezeichnete Prozesse nach auftretenden Semantiken durchsucht und das zu Grunde liegende Modell extrahiert. Als Eingabe dienen dem Verfahren ein Process Execution Log und die Semantikendefinitionen als Regeln, nach welchen das Log durchsucht werden soll. Die Anwendung von Process Mining unter Einbeziehung von Semantikdefinitionen ist in Abbildung 8 dargestellt. Durch die eindeutige Definition der Semantik der Modellierungskonstrukte wird eine Rekonstruktion des zu Grunde liegenden Prozessmodells ermöglicht.

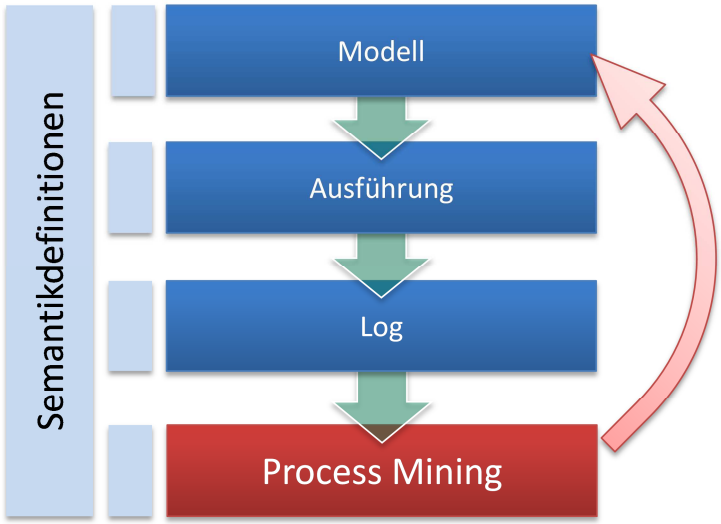

Abbildung 8: Process Mining auf Basis von Semantikdefinitionen

Dabei wird das Process Execution Log zuerst in eine Graph-Datenstruktur überführt. Diese Vorgehensweise ähnelt zunächst dem Ansatz von Agrawal et al. welcher in Abschnitt 3.1.3 erläutert wurde. Der vorgestellte Ansatz geht jedoch über diese grundlegende Analyse hinaus, da nicht nur die Ausführungsreihenfolge der Prozesse sondern auch die beteiligten Prozessdaten betrachtet werden. Die Datenstruktur soll anschließend das Mining nach frei definierbaren Semantiken bzw. Regeln ermöglichen. Da im Grunde für jede definierte Semantik ein eigenes Prozessmodellkonstrukt existieren kann, lassen sich auf diese Weise komplexere Prozessmodelle konstruieren, welche über die Aussagekraft der Nachfolgergraphen von Agrawal et al. hinausgehen.

- Process Mining auf Basis expliziter Semantikdefinitionen *(SemanticPM)* versucht nicht ein imperatives Prozessmodell zu rekonstruieren. Stattdessen werden Prozesse durch semantisch komplexere Modellierungskonstrukte deklarativ modelliert. Die Aussagekraft einer imperativen Modellierung bleibt dabei erhalten, das Modell ist jedoch weniger komplex und einfacher zu generieren.

Im folgenden Abschnitt werden zunächst die Ziele des Ansatzes sowie Voraussetzungen und Einschränkungen erläutert. Anschließend wird die Vorgehensweise dargestellt.

4.1 Zieldefinition und Annahmen

4.1.1 Ziele

In Kapitel 3 wurden bestehende Process Mining Verfahren eingehend dargestellt. Jedes Verfahren basiert auf vordefinierten Modellierungskonstrukten, d.h. der komplette Aufbau der Algorithmen ist abhängig von den vordefinierten Konstrukten. Eine flexible Definition von (anderen) Prozessmodellelementen ist nicht möglich. Das vorrangige Ziel von *SemanticPM* lautet daher:

- SemanticPM generiert eine allgemeingültige Basis-Datenstruktur, welche ein flexibles Process Mining nach beliebig definierten Semantiken ermöglicht.

Es wird ersichtlich, dass die genannten Verfahren stets nur eine Perspektive des Prozesses analysieren. Nach [25] können bestehende Verfahren eindeutig einer Perspektive (Kontrollfluss-, Organisatorische- oder Datenperspektive) zugeordnet werden. Eine perspektivenübergreifende Betrachtung kann somit nicht erfolgen. Erkenntnisse, wie beispielsweise die Abhängigkeit einer Prozessausführungsreihenfolge vom Status der ausführenden Personen, können jedoch nur durch eine kombinierte Betrachtung sämtlicher Perspektiven erfolgen. Daher lässt sich ein weiteres Ziel des vorgestellten Verfahrens formulieren:

- Es wird eine kombinierte Betrachtung unter Einbeziehung sämtlicher Perspektiven der perspektivenorientierten Prozessmodellierung (POPM) innerhalb eines Verfahrens angestrebt und damit die Möglichkeit zum Auffinden von Zusammenhängen zwischen den einzelnen Perspektiven gegeben.

In Abschnitt 3.1.4 wurden der Alpha-Algorithmus und der HeuristicMiner-Algorithmus vorgestellt. Beide Methoden verfahren unterschiedlich mit der Behandlung von Fehlern und Ausnahmen in aufgezeichneten Prozessen. Der Alpha-Algorithmus behandelt Fehler und Ausnahmen wie „normale“ Prozesse, d.h. es erfolgt keine Gewichtung bzw. Kennzeichnung im resultierenden Prozessmodell. Der HeuristicMiner hingegen „filtert“ Ausnahmen und Fehler über Schwellenwerte bei der Analyse des Logs heraus, so dass diese überhaupt nicht im generierten Modell aufzufinden sind. Beide Vorgehensweisen haben Nachteile: Eine Abbildung von Fehlern und Ausnahmen in das Prozessmodell ohne jegliche Kennzeichnung kann irreführend sein und entspricht letztendlich nicht der Realität. Das „Herausfiltern“ von Prozessen, besonders im Fall von Ausnahmen, ist jedoch ebenso unvorteilhaft. Das generierte Prozessmodell kann als Diskussionsgrundlage dienen, weswegen gerade auch Ausnahmen interessant und wichtig erscheinen. *SemanticPM* verfolgt einen Mittelweg: Fehler und Ausnahmen werden betrachtet, fließen jedoch nur gewichtet in das generierte Modell ein. Der Ansatz orientiert sich exakt an dem Verhalten welches im Log beschrieben wird. Es erfolgt keine Filterung über Schwellenwerte. Dieser Vorgehensweise liegt die Closed-World Assumption zu Grunde, da alles was nicht explizit im Log protokolliert wurde als nicht existent und daher als falsch angenommen wird. Ein Ziel des Ansatzes ist wie folgt:

- SemanticPM strebt eine exakte Beschreibung des protokollierten Verhaltens im Process Execution Log nach der Closed-World Annahme unter Einbeziehung von Fehlern und Ausnahmen an.

Der vorliegende Ansatz generiert eine Graph-Datenstruktur welche anhand eines vordefinierten Prozess-Meta-Modells in ein Prozessmodell überführt werden kann. Dieses Modell beschreibt das zu Grunde liegende Log optimal. Ein im Sinne dieser Arbeit optimales Prozessmodell beschreibt einen minimalen und

vollständigen Ergebnisraum. Das generierte Modell muss einen minimalen Ergebnisraum beschreiben, d.h. das Durchlaufen des Modells ermöglicht nur die Erzeugung von Prozessabfolgen wie diese im Log vorkommen (minimal). Andererseits müssen sich jedoch sämtliche im Log vorkommenden Prozessinstanzen mit dem Modell generieren lassen (vollständig). Somit lässt sich das folgende Ziel formulieren:

- Ziel ist eine optimale Beschreibung des Process Execution Logs im Sinne eines minimalen aber vollständig beschriebenen Ergebnisraumes durch ein resultierendes Prozessmodell.

Somit wurden in diesem Abschnitt die Ziele des in dieser Arbeit vorgestellten Process Mining Ansatzes *SemanticPM* definiert. Im folgenden Abschnitt wird auf Voraussetzungen und Einschränkungen, welche dieser Ansatz zu Grunde legt, eingegangen

4.1.2 Voraussetzungen und Einschränkungen

SemanticPM liegen einige Voraussetzungen und Einschränkungen zu Grunde, welche bei dem Mining Vorgang berücksichtigt werden müssen. Diese orientieren sich an den typischen Voraussetzungen, auf denen auch andere Process Mining Algorithmen basieren, welche die Kontrollflussperspektive betrachten.

Anforderungen an die Inhalte des Logs

Um ein erfolgreiches Analysieren des Logs zu ermöglichen, werden auch bei *SemanticPM* die gleichen grundlegenden Anforderungen an das analysierte Log gestellt wie bereits in Abschnitt 3.1.4 dargestellt:

- Jeder Eintrag in der Ereignis-Logdatei bezieht sich auf genau eine Aktivität (d.h. auf ein eindeutig identifizierbares Ereignis eines Prozessschrittes).
- Jeder Eintrag bezieht sich auf genau eine Prozessinstanz.
- Die Einträge besitzen einen Zeitstempel (timestamp) und unterliegen einer Totalordnung.

Injektive Aktivitätszuordnung

Fast alle bekannten Ansätze setzen eine injektive Aktivitätszuordnung voraus, d.h. jeder protokollierte Prozess (oft auch als Aktivität bezeichnet) im Log muss seinen eigenen Namen besitzen. Auch beim vorgestellten Ansatz dürfen keine Duplikate (Duplicate Tasks) im Log existieren.

Zyklische Abläufe (Schleifen)

Der vorliegende Ansatz erlaubt keine Erkennung von zyklischen Abläufen bzw. Schleifen. Eine zukünftige Erweiterung zur Erkennung von Schleifen ist jedoch durchaus möglich. Eine Möglichkeit dies zu erreichen stellt der Lösungsansatz von Agrawal dar. Wiederholte Ausführungen von Aktivitäten werden dabei nummeriert und nach der Mining Prozedur wieder auf eine einzige Aktivität abgebildet.

Unvollständige Daten oder Rauschen

SemanticPM berücksichtigt nicht das Vorhandensein von Rauschen oder die Unvollständigkeit der Information des Logs. Da die Closed-World Annahme zu Grunde liegt, wird explizit die Information modelliert, die auch im Log aufgezeichnet wurde. Somit liegt dem Ansatz eine völlig andere Sichtweise zu Grunde.

4.2 Grundlegende Analyse des Process Execution Logs

Der grundlegende Schritt des Ansatzes ist das erstmalige Durchlaufen des Logs. Das Log besteht aus protokollierten Ereignissen (Events), welche die Start- und Finish-Events von Prozessen (Aktivitäten) markieren. Diese Einteilung der Ereignisse ist [4] entnommen. Von weiteren Typen des Action-Types wird

im Folgenden abstrahiert. Die Daten, welche zu einem Ereignis gespeichert werden sind generell gesehen beliebig. Da sich diese Arbeit jedoch an den Perspektiven des POPM orientiert, wird davon ausgegangen, dass ein Ereignis aus einem 8-Tupel an Prozessdaten besteht. Neben der EventID werden ein eindeutiger Prozessbezeichner (Process ID) sowie eine eindeutige Prozessinstanz ID (d.h. Anwendungsfall ID) aufgezeichnet. An dieser Stelle sei noch einmal an die injektive Aktivitätszuordnung erinnert: Innerhalb einer Prozessinstanz muss jede Process ID eindeutig sein um diese Injektivität zu gewährleisten. Der Action Type gibt an, ob es sich um den Beginn oder das Ende eines Prozessschrittes (einer Aktivität) handelt. Die Spalten Agents (organisationsorientiert), Data (datenorientiert) und Tools (operationsorientiert) sind die Prozessdaten welche der jeweiligen Perspektive des POPM zuzuordnen sind. Der Timestamp wird nur dafür verwendet die Logeinträge in die notwendige Totalordnung zu bringen. Tabelle 8 zeigt ein Beispiel des beschriebenen Logs.

Event ID	Process ID	Case/Instance	Action Type	Agents	Data	Tools	Timestamp
1	**A**	**1**	Start	Agent 1	Rechnung	Word	…
2	**A**	**1**	Finish	Agent 1	Rechnung	Word	
3	**D**	*2*	Start	Boss	Doc 3	Excel	
4	**D**	*2*	Finish	Boss	Doc 3	Excel	
5	**B**	**1**	Start	Agent 2	Doc 2	Word	
6	**C**	*2*	Start	Boss	Doc 2	Excel	
7	**B**	**1**	Finish	Agent 2	Doc 2	Word	
8	**C**	**1**	Start	Agent 4	Doc 3	Word	
9	**C**	*2*	Finish	Boss	Doc 2	Excel	
10	**C**	**1**	Finish	Agent 4	Doc 3	Word	
11	**D**	**1**	Start	…	…	…	
12	**A**	*2*	Start				
13	**A**	*2*	Finish				
14	**B**	*2*	Start				
15	**D**	**1**	Finish				
17	**B**	*2*	Finish				

Tabelle 8: Beispiel eines Logs mit zu Grunde liegenden Prozessdaten

Dieser Ausschnitt eines Logs wird in den folgenden Abschnitten als laufendes Beispiel verwendet und dient zu Erläuterungszwecken.

4.2.1 Klassifikation von Parallelitäten und Nachfolgebeziehungen

Der nächste Schritt zur Analyse des Logs ist die Aufspaltung der aufgezeichneten Ereignisse nach einzelnen Prozessinstanzen bzw. Prozessanwendungsfällen. Hierfür werden separate Listen für jede Instanz angelegt und die einzelnen Ereignisse anhand des Feldes „Case/Instance“ der jeweiligen Liste zugeteilt. Die Prozessdaten sowie der Action Type werden dabei in die Listen übernommen. Abbildung 9 zeigt einen Ausschnitt aus der entstehenden Aufteilung der Logereignisse aus Tabelle 8.

Instanz/Case 1		
Process ID	**Action Type**	**…**
A	start	…
A	Finish	…
B	start	…
B	finish	…
…		

Instanz/Case 2		
Process ID	**Action Type**	**…**
D	start	…
D	finish	…
C	start	…
C	finish	…
…		

Abbildung 9: Aufteilung der Logeinträge nach Prozessinstanzen

Anhand der nun vorliegenden Listen kann eine Klassifizierung der Beziehung zwischen einzelnen Prozessen bezgl. deren zeitlicher Abarbeitungsreihenfolge erfolgen. Die Klassifizierung basiert auf den Kombinationen des Feldes „Action Type“ zweier aufeinander folgenden Ereignisse. Grundlegend wird zwischen einer parallelen Ausführung und einer sequentiellen Ausführung unterschieden. Die verschiedenen Arten der Beziehungen sind in Abbildung 10 dargestellt und werden nun erläutert.

Parallelitäten

Es existieren zwei Typen von Parallelitäten innerhalb des Logs. Betrachte man beispielsweise die Start/Finish-Ereignisse zweier Aktivitäten A und B. In der ersten möglichen Ereignisreihenfolge folgt auf den Start der Aktivität A (Start A) der Start von Aktivität B (Start B), anschließend das Ereignis Finish A und zuletzt Finish B (Abbildung 10, Punkt 1). Im Log sind somit die Events von A und B verschränkt. Daneben existiert eine zweite mögliche Ereignisreihenfolge in der die Ereignisse von A die Ereignisse von B umschließen. Demnach wäre die Reihenfolge also: Start A, Start B, Finish B, Finish A (Abbildung 10, Punkt 2). Beide dargestellten Ereignisreihenfolgen stellen Parallelitäten im Sinne der Ausführungsreihenfolge dar, da sich die Zeiträume der Ausführung der Aktivitäten A und B teilweise (1) oder in Bezug auf die Aktivität B sogar vollständig überlappen (2).

Direkte Nachfolger

Neben Parallelitäten existiert eine direkte Nachfolger Beziehung. Diese gibt an, dass das Start Ereignis einer Aktivität D im Log direkt auf das Finish-Ereignis einer Aktivität C folgt (Abbildung 10, Punkt 3). Eine mögliche Ereignisreihenfolge, welche zur Entdeckung einer direkten Nachfolger Beziehung führen würde, wäre demnach Start C, Finish C, Start D, Finish D.

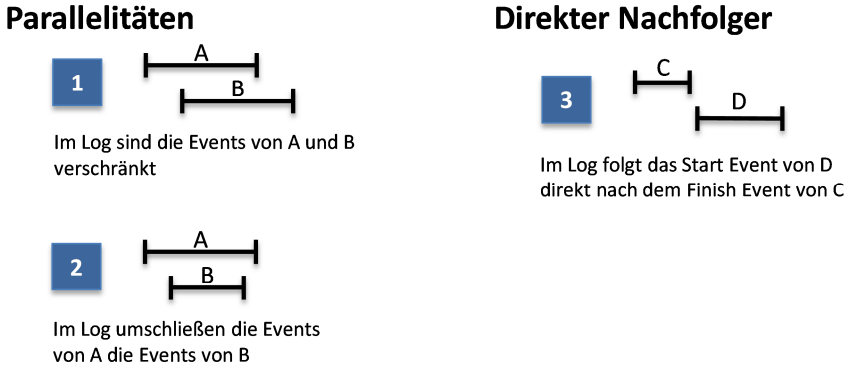

Abbildung 10: Verschiedene Arten von Parallelitäts- und Nachfolgerbeziehungen

Anhand der in dieser Weise erfolgten Klassifizierung der Beziehungen zwischen Aktivitäten kann nun der Aufbau von sog. *Instanzgraphen* erfolgen.

4.2.2 Aufbau von Instanzgraphen

Die Information, wie sich zwei Prozesse zueinander verhalten (d.h. ob sie parallel ausgeführt werden können oder der eine von dem anderen abhängt) kann nun gebündelt mit den zugeordneten Daten der betreffenden Prozesse in einer Graph-Datenstruktur dargestellt werden. Es bieten sich hierfür nach einer oberflächlichen Betrachtung zwei Möglichkeiten: Generierung eines gemeinsamen Graphen für alle Instanzen oder Generierung separater Graphen für jede Instanz. Der Entscheidung für eine der beiden Möglichkeiten liegt zum ersten Mal der Gedanke der perspektivenübergreifenden Betrachtung der aufgezeichneten Information zu Grunde.

4.2.2.1 Gemeinsamer Graph für alle Instanzen

Bei dieser ersten Herangehensweise werden sämtliche Informationen und Beziehungen, welche durch die Klassifizierung aus Punkt 4.2.1 gewonnen wurden, in einen einzigen Graphen eingetragen. Es ist zu beachten, dass sich die Kontrollfluss-Beziehungen (d.h. Parallelität oder Nachfolger) und die gespeicherten Prozessdaten der Prozessschritte von Instanz zu Instanz unterscheiden können. Wird beispielsweise in einer Instanz eine direkte Nachfolger Beziehung von A zu C entdeckt und in einer anderen Instanz eine direkte Nachfolgerbeziehung von C zu A, so sind beide Beziehungen im resultierenden Graphen eingezeichnet. Außerdem werden die aufgezeichneten Prozessdaten akkumuliert, womit deren Bezug zur Ausführungsreihenfolge verloren geht. Dieses Beispiel ist in Abbildung 11 illustriert. Die Pfeile des Graphen stellen direkte Nachfolger Beziehungen zwischen den Prozessschritten (Knoten) dar. Die Prozessdaten bezgl. jedes Prozesses „haften“ an den Knoten. Betrachtet man nun die Prozesse A und C, so zeigt sich, dass in verschiedenen Instanzen sowohl einmal C ein Nachfolger von A war als auch vice versa. Die Akkumulation der Prozessdaten der organisatorischen Perspektive (ausführende Agenten) zeigt sich bei Prozess C. Einmal wurde die Aktivität von einem Boss ausgeführt, in einem anderen Anwendungsfall von einem Agent Nr. 4. Innerhalb des gemeinsamen Graphen sind die Agenten akkumuliert und ohne Bezug zu einem bestimmten Anwendungsfall.

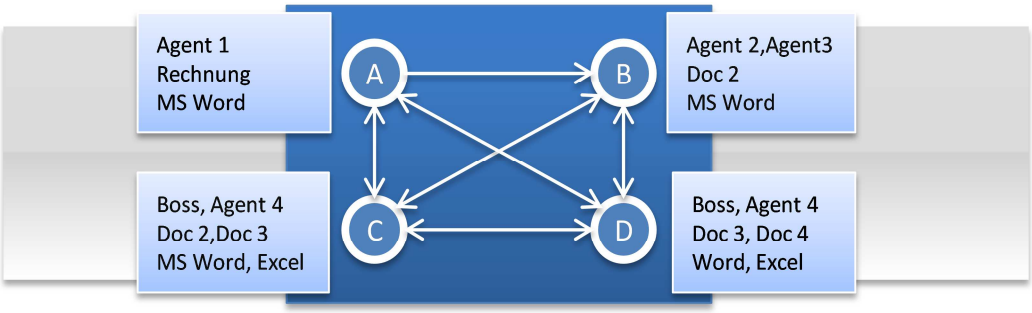

Abbildung 11: Gemeinsamer Graph für alle Instanzen

Durch die erläuterte Akkumulation der Beziehungen und der Prozessdaten geht Information verloren. Es lassen sich eine Reihe von Fragen formulieren, welche durch Betrachtung des Graphen nicht mehr beantwortet werden können. Exemplarisch genannt seien:

- Ist Agent 2 immer an Prozess B beteiligt?
- Muss Agent 4 in Prozess C das Tool „Word“ wegen dem Dokument Doc 2 oder Doc 3 benutzen?
- Ist die Reihenfolge der Prozesse C und D immer egal oder nur wenn der Boss die Aktivitäten ausführt?

Es wird ersichtlich, dass ein gemeinsamer Graph für alle Instanzen, gerade aufgrund der perspektivenübergreifenden Fragestellungen welche *SemanticPM* betrachtet, keine adäquate Grundlage für das Mining nach frei definierbaren Semantiken bzw. Regeln bietet.

4.2.2.2 Separate Graphen für jede Prozessinstanz

Die zweite Möglichkeit zur Darstellung der Information sind separate Graphen pro Prozessinstanz. Es wird für jeden Anwendungsfall, d.h. für jede Prozessinstanz des Logs, ein eigener Graph (sog. *Instanzgraph*) erzeugt. Dabei werden wie zuvor die auftretenden Prozessschritte durch Knoten repräsentiert, wobei die Prozessdaten, wie Agenten, Daten und Tools, als Felder und Listen hinterlegt sind. Die Kanten zwischen den Prozessen besitzen zwei Informationen: Zum einen den Typ der Beziehung (Nachfolger oder Parallelität), zum anderen ob es sich um einen direkten oder um einen transitiven Nachfolger handelt. Ein Instanzgraph wird wie folgt generiert:

1) Durchlaufen der instanzspezifischen Ereignisliste X und Klassifizierung der Beziehungen zwischen den Prozessen.
2) Für jeden auftretenden Prozess wird ein Knoten innerhalb des Instanzgraphen X erzeugt und die aufgezeichneten zugehörigen Prozessdaten angehängt.
3) Für jede direkte Nachfolger Beziehung (B folgt im Log direkt auf A) wird eine Kante A → B vom Typ *successor*, *direct* im Graph erzeugt. Des Weiteren wird für jede Parallelitätsbeziehung (C und D werden im Log parallel ausgeführt) eine Kante C → D vom Typ *Parallel*, *direct* im Graph erzeugt.
4) Für jede direkte Nachfolger Beziehung im bestehenden Graph wird der transitive Abschluss gebildet, d.h. existieren bereits die Kanten A → B (Typ: *successor*, *direct*) sowie B → C (Typ: *successor*, *direct*), so wird die Kante A → C (Typ: *successor*, *transitive*) im Instanzgraph X erzeugt.

Es ist zu beachten, dass für Parallelitäten im Allgemeinen keine Transitivität gilt, d.h. es kann nicht automatisch gefolgert werden, dass eine Parallelität zwischen A und B sowie B und C auch eine Parallelität zwischen A und C zur Folge hat. In Abbildung 12 sind drei Instanzgraphen, welche aus der Analyse des Beispiellogs aus Tabelle 8 hervorgehen, dargestellt.

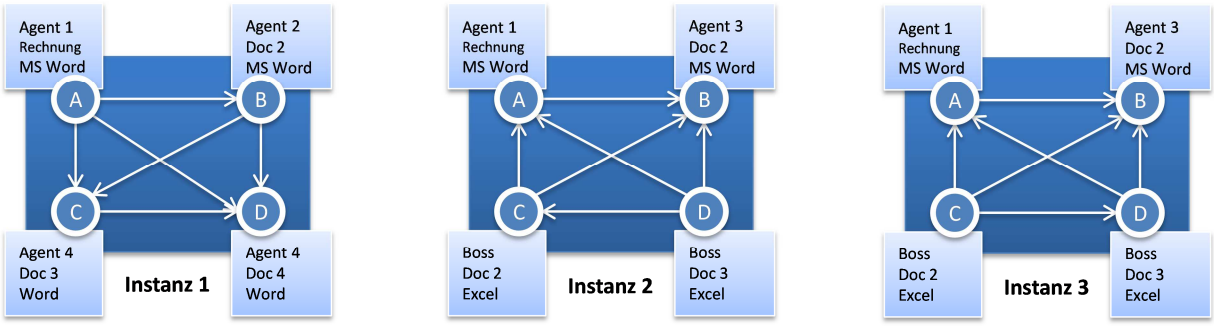

Abbildung 12: Separate Graphen für jede Prozessinstanz (Instanzgraphen)

Es wird ersichtlich, dass bei separater Betrachtung jeder Prozessinstanz auf Zusammenhänge zwischen den Perspektiven geschlossen werden kann. Dies wird anhand des folgenden Beispiels verdeutlicht.

Beispiel 1

Betrachte man die Prozesse C und D. Innerhalb der Instanz 1 folgt D auf C, wobei die ausführende Person beider Schritte „Agent 4“ ist. Innerhalb der Instanzen 2 und 3 werden die Schritte C und D von einem „Boss“ ausgeführt, d.h. einer in der Organisationsstruktur höher gestellten Person. Hier scheint die Ausführungsreihenfolge der Schritte C und D egal zu sein, da in Instanz 2 der Schritt C auf D folgt, in Instanz 3 jedoch D auf C. Es liegt nahe, dass die Ausführungsreihenfolge der Schritte C und D daher abhängig von den ausführenden Agenten entweder fest geordnet (im Fall des „normalen“ Mitarbeiters Agent 4) oder beliebig ist (im Fall des „Boss“ Mitarbeiters). Diese Schlussfolgerung lässt sich nur aus einer separaten Betrachtung der Instanzen ziehen, nicht jedoch aus dem gemeinsamen, akkumulierten Graphen aus Abschnitt 4.2.2.1.

- Bei der genannten Schlussfolgerung, d.h. dass die Ausführungsreihenfolge von C und D von den ausführenden Agenten abhängig ist, liegt die Closed-World Annahme zu Grunde. Das Log enthält keine Instanz in der auch bei Ausführung durch „Agent 4“ die Reihenfolge umgekehrt ist und C auf D folgen würde. Deshalb wird bei der Folgerung davon ausgegangen, dass sich die Situation stets in der aufgezeichneten Weise verhält.

Im folgenden Abschnitt wird das Vorgehen zur Schlussfolgerung von Zusammenhängen formalisiert.

4.3 Semantische Interpretation der Instanzgraphen

Das Vorgehen zur Schlussfolgerung von Zusammenhängen, d.h. das Auffinden von vordefinierten Semantiken, wird als *semantische Interpretation* bezeichnet. Die semantische Interpretation setzt sich aus drei Schritten zusammen:

1) Aufsuchen von Semantiken unter Einbeziehung der Instanzgraphen (Abschnitt 4.3.1)
2) Generierung eines einzigen Graphen welcher sämtliche Zusammenhänge bzw. Semantiken zwischen Prozessen und Eigenschaften von Prozessen enthält (Abschnitt 4.3.2)
3) Kapselung von Prozessen, d.h. separate Betrachtung der funktionalen Perspektive (Abschnitt 4.3.3)

4.3.1 Algorithmische Umsetzung der semantischen Interpretation

4.3.1.1 Grundlegende Vorgehensweise und Einteilung von Semantikdefinitionen

Der erste Schritt um das Auffinden von vordefinierten Semantiken (d. h. Regeln) zu ermöglichen, ist die Analyse des Aufbaus und die Einteilung der Regeln. Wie bereits in Abschnitt 2.3.2 erläutert wurde, verwendet diese Arbeit die SWRL um Semantikdefinitionen zu definieren und darzustellen. Dementsprechend besitzen Regeln eine linke und eine rechte Seite (vgl. Abbildung 13). Die linke Seite der Regel enthält sämtliche Bedingungen welche erfüllt sein müssen, so dass die rechte Seite der Regel gefolgert werden kann. Des Weiteren stellt die linke Seite der Regel einen Indikator für das Modellierungskonstrukt dar, welches durch die Regel repräsentiert wird. Die rechte Seite der Regel liefert neben der Konsequenz, welche es ebenso zu prüfen gilt, einen Indikator für die Perspektive der die aktuell betrachtete Regel zuzuordnen ist. Handelt es sich beispielsweise um eine Regel der verhaltensorientierten Perspektive, so treten auf der rechten Seite der Regel Variablen von Prozessen auf, die in eine zeitliche Beziehung („P1 beginnt nach dem Ende von P2“, startedAfterFinished(P1, P2)) gesetzt werden. Handelt es sich um eine Regel der organisatorischen Perspektive, so treten Variablen auf, die Agenten bzw. ausführende Personen beschreiben.

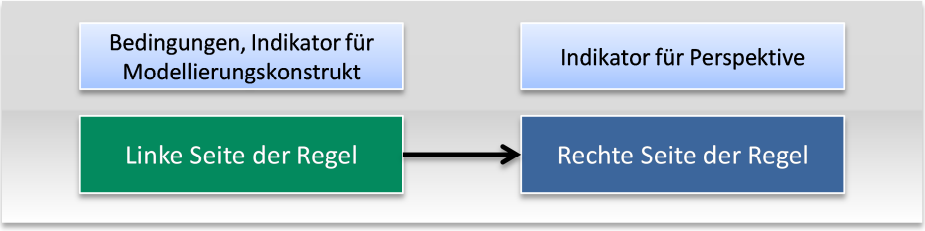

Abbildung 13: Aufbau einer Semantikdefinition

Beispiel 2

Als laufendes Beispiel einer Semantikdefinition wird, angelehnt an das Beispiel aus Abschnitt 4.2.2.2, eine Regel verwendet die das Modellierungskonstrukt *BossConnector* beschreibt und der verhaltensorientierten Perspektive zuzuordnen ist. Informell beschrieben definiert dieses Konstrukt genau die Semantik, welche zuvor in Beispiel 1 dargestellt wurde. Sind zwei Prozesse C und D durch den *BossConnector* (C → D) verbunden und sind die ausführenden Personen beider Prozesse Mitglieder einer Klasse *NotBoss* (d. h. sie sind „normale“ Mitarbeiter), so ist deren Reihenfolge festgelegt und C muss stets vor dem Start von D beendet werden. Andererseits impliziert diese Regel auch, dass für Agenten die nicht Mitglieder der Klasse *NotBoss* sind, keine Einschränkung (*Constraint*) bezgl. der Reihenfolge von C und D zu gelten hat. Die formale Schreibweise dieser Regel in SWRL lautet wie folgt:

```
BossConnector(?BC) & From(?BC, ?P1) & To(?BC, ?P2) & hasAgent(?P1, ?A) &
NotBoss(?A) & hasAgent(?P2, ?B) & NotBoss(?B)  ⇨  startedAfterFinished(?P2, ?P1)
```

Beispiel 3

Ein weiteres Modellierungskonstrukt, der *StrictControlFlowConnector* [28] ist ebenfalls der verhaltensorientierten Perspektive zuzuordnen. Sind zwei Prozesse A und B durch den *StrictControlFlowConnector* (A → B) verbunden, so ist deren Reihenfolge in allen Anwendungsfällen festgelegt und A muss stets vor dem Start von B beendet werden. Hier ist es folglich irrelevant, welche Agenten beteiligt sind, die Abhängigkeit gilt stets und für jede Instanz. Die Regel in SWRL lautet:

```
StrictControlFlowConnector(?SC) & From(?SC, ?P1) & To(?SC, ?P2)  ⇨
                    startedAfterFinished(?P2, ?P1)
```

Die Variablen ?P1 und ?P2 stellen die beteiligten Prozesse dar. Es wird ersichtlich, dass auf der rechten Seite der *BossConnector*- und der *StrictControlFlowConnector*-Regel zwei Variablen verwendet werden. Die Anzahl der auftretenden Variablen auf der rechten Seite einer Regel ist grundlegend für die Art des Modellierungskonstrukts und erfordert die Definition einiger Begriffe.

Wertigkeit

Der Begriff *Wertigkeit* beschreibt die Anzahl der Variablen der rechten Seite einer Regel. Die Wertigkeit ist ein Indikator für die Art der Semantik, welche durch eine Regel beschrieben wird. Einerseits kann es sich um eine Eigenschaft eines einzigen Prozesses handeln, anderseits um eine Beziehung zwischen Prozessen.

Attribute

Einwertige Regeln (d.h. die rechte Seite der Regel besitzt nur eine Variable) liefern sog. *Attributes*. Attributes repräsentieren Eigenschaften einzelner Prozesse und keine Beziehung zu einem anderen Prozess.

Beispiel: `Process(?P1) & hasAgent(?P1, ?A)  & hasTool(?P1, ?T1) & hasName(?T1, „ProE“) ⇨  AgentClass(?A, „Engineer“)`

Edge

Zweiwertige Regeln liefern *Edges* zwischen zwei Prozessen. Edges repräsentieren Beziehungen/Relationen zwischen zwei Prozessen.

Beispiel: *BossConnector*

Zur Verdeutlichung der verschiedenen Begriffe folgen noch einige Beispiele von rechten Seiten entsprechender Regeln:

`startedAfterFinished(?P1, ?P2)` → *Verhaltensorientiert, zweiwertig*
`sameAs(?Agent1, ?Agent2)` → *Organisationsorientiert, zweiwertig*
`sameAs(?Doc1, ?Doc2)` → *Datenorientiert, zweiwertig*
`Tool(?T1) & hasName(?T1, „Latex“)` → *Operationsorientiert, einwertig*

Somit wurden der Aufbau und die Einteilung von Semantikdefinitionen definiert. Im folgenden Abschnitt wird nun die algorithmische Vorgehensweise zum Auffinden einer Semantikdefinition unter Einbeziehung der Instanzgraphen erläutert.

4.3.1.2 Algorithmischer Beweis der Gültigkeit einer Semantikdefinition

Das Auffinden der Semantik eines Prozesses bzw. zwischen zwei Prozessen stellt den Kern von *SemanticPM* dar. Hierfür werden alle vorhandenen Instanzgraphen durchlaufen, die aktuell betrachteten Prozesse gesucht und deren Eigenschaften und Prozessdaten analysiert. Da generell zwischen jeder Kombination von zwei Prozessen x und y eine Beziehung bestehen kann, müssen auch alle Kombinationen (x, y) für jede zu prüfende (zweiwertige) Regel betrachtet werden. Der Beweis der Gültigkeit einer zweiwertigen Regel zwischen Prozessen x und y hat nun folgenden Aufbau (vgl. Abbildung 14):

- Zuerst wird angenommen, dass die Beziehung zwischen den zwei aktuell betrachteten Prozessen existiert (`proof = true`). Bei der Beweisführung handelt es sich folglich um einen Widerspruchsbeweis.

- Anschließend werden die beteiligten Prozesse auf die Erfüllung der nötigen Bedingungen hin überprüft. Das bedeutet die Prozesse und deren zugeordnete Prozessdaten müssen die Klassenzugehörigkeiten und Properties der linken Seite der Regel erfüllen. Kann eine Information nicht ohne weiteres aus den Instanzgraphen extrahiert werden, muss potentiell eine angebundene Wissensbasis (beispielsweise eine Ontologie) befragt werden. In Form von Hilfsmethoden wird über eine ID weitergehende Information abgefragt. Betrachtet man das Beispiel aus Abbildung 14, so kann beispielsweise nicht aus den Instanzgraphen geschlossen werden ob es sich bei dem ausführenden Agenten von Prozess x um einen Boss handelt oder nicht. Dementsprechend wird in einer Hilfsmethode `NotBoss(x.Agent)` eine Verbindung zu einer Ontologie aufgebaut und daraus extrahiert, ob es sich bei jenem Agenten um einen Boss handelt oder nicht.

- Sind sämtliche Bedingungen erfüllt, wird geprüft ob innerhalb des betrachteten Instanzgraphen auch die Konsequenz (d.h. die rechte Seite der Regel) erfüllt ist. Hierfür wird die rechte Seite der Regel negiert. Wird die Negation der Konsequenz im betrachteten Instanzgraphen aufgefunden, so wurde die Gültigkeit der Regel zwischen den Prozessen x und y widerlegt (`proof = false`).

- Wurde innerhalb der Liste der Instanzgraphen kein Gegenbeispiel gefunden (d.h. `proof` ist nach Betrachtung aller Instanzgraphen weiterhin `true`), so wurde die Semantik bewiesen und wird als gültig zwischen den Prozessen x und y deklariert. (*BossConnector(x → y)*)

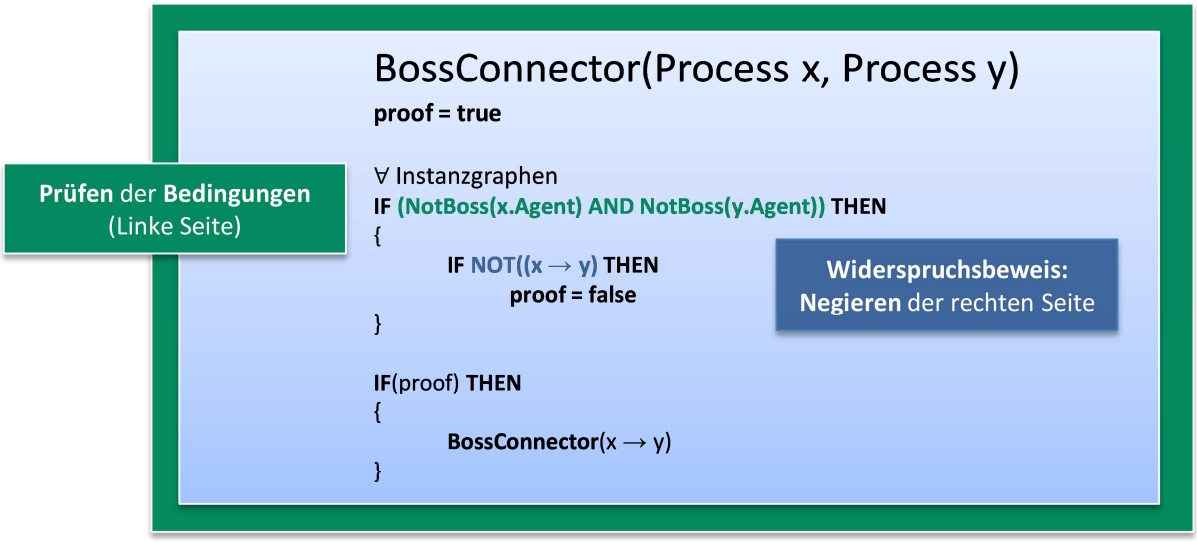

Abbildung 14: Algorithmus zum Beweis der *BossConnector* Semantik

Betrachtet man noch einmal die linke Seite der Definition der *BossConnector* Semantik aus Beispiel 2, so fällt auf, dass Teile wie die Klassenzugehörigkeit `From(?BC, ?P1)` und `To(?BC, ?P1)` innerhalb der Prüfung der nötigen Bedingungen keinen Eingang finden. Die Klassen *From* und *To* geben die Richtung des Modellierungskonstrukts *BossConnector* an. Diese Richtung ist aber bei der Prüfung bereits dadurch gegeben, mit welcher Kombination der Prozesse x und y der Algorithmus aufgerufen wird. Wird der Algorithmus mit der Parameterkombination „*Process x, Process y*“ aufgerufen, wird die Gültigkeit in Richtung x → y überprüft. Ist die Parameterkombination jedoch „*Process y, Process x*“, wird die Gültigkeit in Richtung y → x getestet. Somit sind Klassenzugehörigkeiten zu *From* und zu *To* bereits vorab geprüft und fließen nicht mehr in die Bedingungen ein. Selbiges gilt für das Property *hasAgent*. Ist dem betrachteten Prozess kein Agent zugeordnet, wäre die Betrachtung der *BossConnector*-Semantik obsolet.

- *SemanticPM* liegt die Closed-World-Annahme zu Grunde. Gerade deshalb kann bei der semantischen Interpretation ein Widerspruchsbeweis angewendet werden. Wird in den Instanzgraphen (diese stellen die vorhandene und verarbeitete Wissensbasis dar) kein Gegenbeispiel gefunden, so wird die Annahme als gültig angesehen.

- Der Aufbau der semantischen Interpretation ist für einwertige Regeln identisch zu zweiwertigen Regeln. Es wird jedoch lediglich ein Prozess der Funktion übergeben.

4.3.1.3 Abarbeitungsreihenfolge der (Teil-)Algorithmen

Da es im Allgemeinen mehrere Regeln und somit Modellierungskonstrukte pro Perspektive geben kann, stellt sich, vor allem im Bereich der verhaltensorientierten Perspektive die Frage, in welcher Reihenfolge die Prüfung der Semantiken abgearbeitet werden soll. Betrachtet man die Beispiele 2 und 3 so wird ersichtlich, dass vor der Überprüfung auf eine *BossConnector*-Semantik die Überprüfung auf eine *StrictControlFlowConnector*-Semantik erfolgen muss. Der *StrictControlFlowConnector* besitzt keine Bedingungen auf der linken Seite, d.h. für jede Prozessinstanz wird die Konsequenz (rechte Seite der Regel) überprüft. Beim Auffinden des *BossConnectors* wird nicht unbedingt jede Instanz auf die Gültigkeit der Konsequenz hin überprüft, da einige Instanzen aufgrund der Nichterfüllung der Bedingung ausgelassen werden. Die Menge, die für den *StrictControlFlowConnector* zu prüfen ist, ist also *mächtiger* als für den *BossConnector*, da er eine größere Menge an zu betrachtenden Fällen einschließt. Die

Reihenfolge der abzuarbeitenden Regeln richtet sich also nach der Menge an Prozessinstanzen (Anwendungsfällen) deren Konsequenz zu überprüfen ist. Prägnant formuliert gilt folgende Regel:

Allgemeine Regel (Große Menge zu prüfen) **vor** **Spezieller Regel** (Kleine Menge zu prüfen)

Im Allgemeinen ist die Vorhersage, welche Regel allgemeiner und welche Regel spezieller ist, nicht einfach zu treffen. Betrachtet man zwei Regeln, welche verschiedene Bedingungen auf der linken Seite vorweisen, so ist nicht von Anfang an klar, wie umfangreich die durch die Bedingungen hervorgerufenen Constraints den Lösungsraum einschränken und somit wie viele Instanzen die Bedingungen erfüllen und zur Prüfung der Konsequenz gelangen. Die Ordnung der Reihenfolge erfordert daher Fachwissen und tiefgehendes Verständnis der betrachteten Domäne. Abbildung 15 illustriert die Problematik. Es kann nicht vorhergesagt werden, auf welche Weise die Bedingungen einer Regel den Lösungsraum einschränken. Die Bedingung 1 *hasName(?A, „Jack“)* schränkt den Lösungsraum beispielsweise sehr ein, da wahrscheinlich nur eine geringe Anzahl von Prozessen von „Jack“ ausgeführt wurden. Bedingung 2 hingegen schränkt den Lösungsraum nur wenig ein, da fast jeder Prozess eine ausführende oder verantwortliche Person besitzt (es sei denn es wurde bei der Protokollierung vergessen).

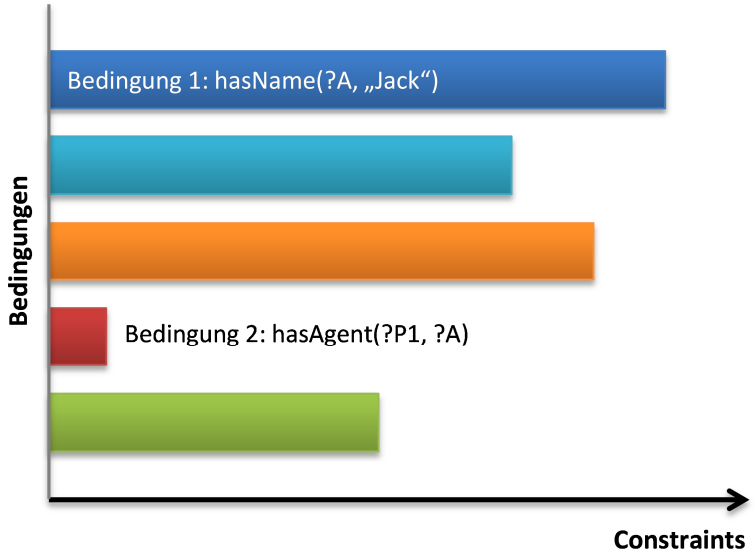

Abbildung 15: Verschiedene Arten von Bedingungen erzeugen unterschiedliche Constraints

Diejenigen Regeln, deren Bedingungen den Lösungsraum nur gering einschränken, werden daher zuerst behandelt. Beim *StrictControlFlowConnector* wird der Lösungsraum aufgrund fehlender Bedingungen überhaupt nicht eingeschränkt, weshalb er auch zuerst geprüft wird.

4.3.2 Generierung des Semantikgraphen

Während bzw. nach der Abarbeitung der Algorithmen zum Beweis der Semantiken wird ein Graph aufgebaut, welcher sämtliche Eigenschaften und Zusammenhänge der Prozesse darstellt. Dieser Graph heißt *Semantikgraph*. In diesem Graph stellen die Knoten wiederum die auftretenden Prozessschritte dar, während Kanten Beziehungen zwischen den Prozessen signalisieren, d.h. die in Abschnitt 4.3.1.1 definierten *Edges*. Knoten besitzen wiederum Felder für Prozessdaten, jedoch außerdem eine Liste an den ebenfalls in Abschnitt 4.3.1.1 definierten *Attributes*. Eine Kante besitzt Felder für den Edge-Typ, d.h. für die Art der Semantik zwischen den Prozessen, sowie wiederum ein Feld welches signalisiert, ob es sich um eine direkte oder eine transitive Beziehung handelt. Wurde eine Regel für einen Prozess x bzw. für eine Kombination x, y bewiesen, so wird entweder dem Prozessknoten ein entsprechendes Attribut

(einwertige Regel), oder eine Edge mit entsprechendem Edge-Typ (zweiwertige Regel) zwischen den beiden Prozessen hinzugefügt. In Abbildung 16 ist die Vorgehensweise zur Generierung des Semantikgraphen für zweiwertige Regeln zusammenfassend und exemplarisch dargestellt.

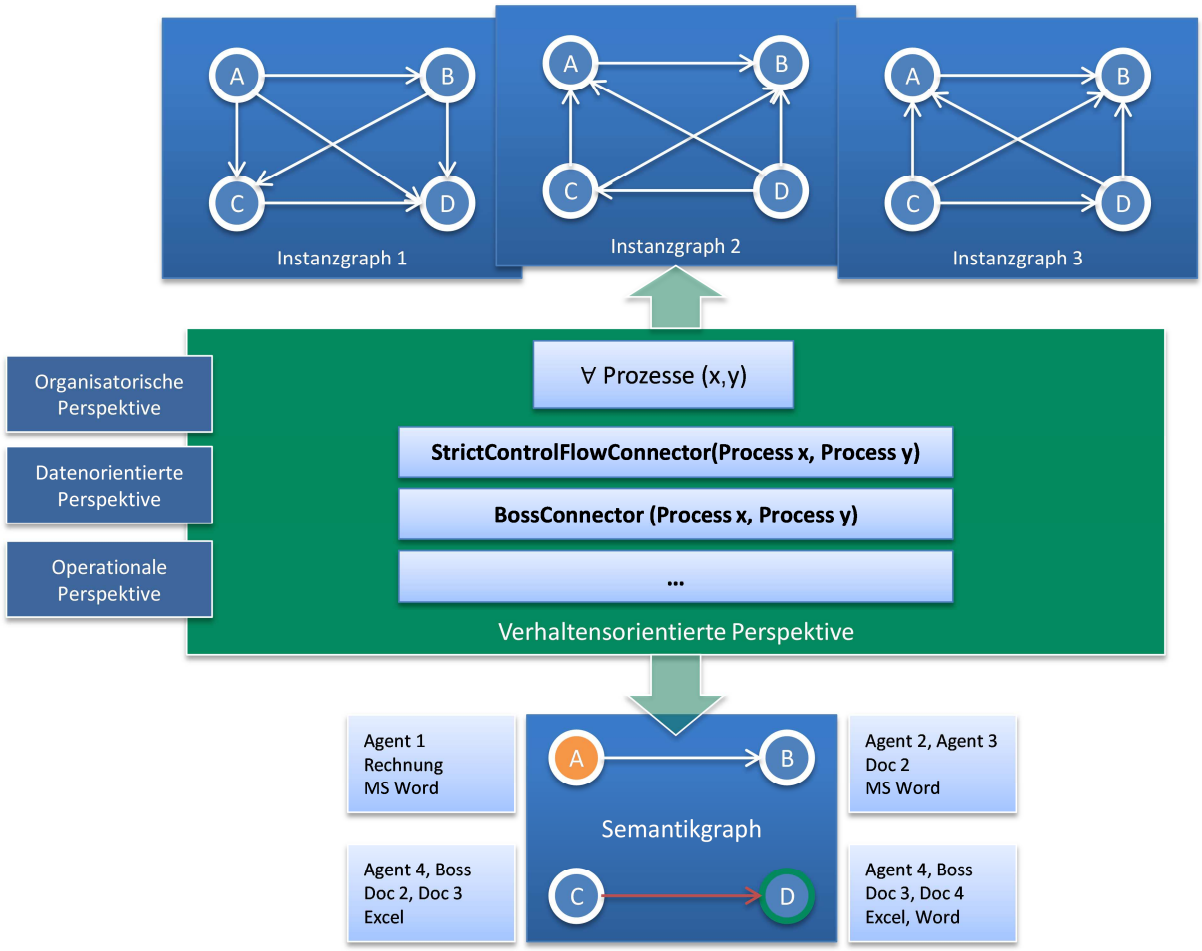

Abbildung 16: Zusammenfassende Vorgehensweise zur Generierung des Semantikgraphen

Die Grafik zeigt, dass sämtlichen Semantikbeweisen die Menge an Instanzgraphen als Input zu Grunde liegt. Innerhalb des grünen Bereichs wird für jede Prozesskombination x, y das Suchen und Beweisen von Regeln durchgeführt. Dargestellt sind nur exemplarische Regeln der verhaltensorientierten Perspektive, jedoch werden in diesem Bereich alle Perspektiven betrachtet und entsprechende Attributes bzw. Edges im Semantikgraph generiert. Der resultierende, vollständige Semantikgraph enthält alle Relationen zwischen Prozessschritten welche auf Basis der Closed-World Annahme bewiesen werden konnten, sowie sämtliche Prozessdaten.

4.3.3 Kapselung von Prozessen

In den vorherigen Abschnitten wurde der Semantikgraph erzeugt. Bei dessen Generierung wurden sämtliche Perspektiven des POPM mit Ausnahme der funktionalen Perspektive kombiniert betrachtet und Zusammenhänge extrahiert. Im Folgenden wird die funktionale Perspektive anhand des Semantikgraphen analysiert. Wie in Abschnitt 2.2 definiert wurde, deklariert diese Perspektive die funktionalen Einheiten. Es gilt zu unterscheiden zwischen elementaren und kompositen Prozessen, die sich wiederum aus anderen Prozessen – sowohl elementar als auch komposit – zusammensetzen. Welche Prozesse (Subprozesse) in einem kompositen Prozess (dieser fungiert als „Kapsel“ für die untergeordneten Subprozesse) enthalten sind, wird dabei nur auf Grundlage gemeinsamer Eigenschaften in Bezug auf den

Kontrollfluss definiert. Die funktionale Perspektive basiert somit auf der verhaltensorientierten Perspektive. In den folgenden Abschnitten wird unter einer *Kapsel* ein Prozess verstanden, welcher wiederum aus anderen Prozessen (Subprozessen) besteht. Die Kapseln werden dabei nur durch eine ID identifiziert. Wird eine Kapsel explizit durch einen Namen ausgezeichnet, so wird aus der Kapsel ein ausgezeichneter, kompositer Prozess. Der Kapselung von Prozessen liegen folgende Regeln zu Grunde [28]:

```
Process(?P1) & Process(?P2) & MemberOf(?P1, ?P2) & StartedAfterFinished(?P2, ?P)  ⇨
                          startedAfterFinished(?P1, ?P)

Process(?P1) & Process(?P2) & MemberOf(?P1, ?P2) & StartedAfterFinished(?P, ?P2)  ⇨
                          startedAfterFinished(?P, ?P1)
```

Informell beschrieben besagen die obigen Regeln, dass ein Prozess P, welcher nach (bzw. vor) einem kompositen Prozess P2 gestartet (bzw. beendet) wird, auch nach (bzw. vor) einem in Prozess P2 enthaltenen Subprozess P1 gestartet (bzw. beendet) werden muss. Da bei der Prozessausführung nur elementare Prozesse ausführbar sind und deshalb im Log auch nur elementare Prozesse aufgezeichnet sind, sind im Allgemeinen bei Vorliegen eines Logs keine kompositen Prozesse (Kapseln) bekannt.
Eine Prüfung der obigen Regeln anhand einer Beweisführung wie in Abschnitt 4.3.1 ist aus drei Gründen nicht möglich:

1) Um den vollständigen Beweis für eine Enthaltensein-Relation (*MemberOf*) erbringen zu können, müssen obige Regeln *gleichzeitig* gelten.

2) Beide Regeln müssen *für alle* Prozesse P gelten, welche im Semantikgraph eine *StartedAfterFinished*-Relation zu Prozess P1 besitzen.

3) Beide Regeln müssen evtl. mehrmals überprüft werden, da Prozesshierarchien aufgebaut werden sollen. Eine einmalige Überprüfung würde nur die elementaren Prozesse in der ersten Ebene klassifizieren und keine MemberOf-Hierarchien aufbauen.

Aus diesen Gründen wird zur Generierung von Kapseln und somit Prozesshierarchien folgende Vorgehensweise verwendet:

1) Da Kapseln auf Basis einer *StartedAfterFinished*-Semantik generiert werden, werden nur die Kontrollfluss-Semantiken innerhalb des Semantikgraphen betrachtet. Zu beachten ist nun, dass die Bedingungen der Kontrollfluss-Semantiken für die Generierung der Kapseln nicht relevant sind, d.h. dass alle definierten Semantiken der verhaltensorientierten Perspektive bei der Suche nach Kapseln als semantisch gleich betrachtet werden. Anschaulich gesehen enthält der für die Analyse der funktionalen Perspektive angepasste Semantikgraph nun nur noch Kanten, welche eine *StartedAfterFinished*-Relation zwischen zwei Prozessen X und Y darstellen. Dies ist auch nachvollziehbar, da die funktionale Perspektive wie gesagt nur auf der verhaltensorientierten Perspektive basiert (anhand der obigen Regeln ersichtlich) und somit Elemente anderer Perspektiven (z.B. Bedingungen welche die organisatorische Perspektive betreffen, vgl. *BossConnector*) nicht berücksichtigt werden müssen.

2) Nach Reduzierung des Semantikgraphen auf die notwenige Information wird ein hierarchisches Verfahren nach dem Bottom-Up-Prinzip eingeleitet. Für je zwei Prozesse aus dem Semantikgraphen wird anhand derer gemeinsamen Beziehungen entschieden, ob sie in der

gleichen Kapsel enthalten sind. Zwei Prozesse sind genau dann in einer gemeinsamen Kapsel, wenn sie im Semantikgraph die gleichen Vorgänger als auch die gleichen Nachfolger besitzen. Formal geschrieben lautet diese Regel wie folgt:

```
IF   ∀(P | (startedAfterFinished(P1, P)) ⇨ startedAfterFinished(P2, P)) AND
     ∀(P | startedAfterFinished(P, P1)) ⇨ startedAfterFinished(P, P2))
       THEN   MemberOf(P1, Capsule1) AND MemberOf(P2, Capsule1)
```

3) Da sich eine Kapsel wiederum aus Kapseln zusammensetzen kann, wird nach dem ersten Durchlauf, sofern mehrere Kapseln erzeugt wurden, wiederum für je zwei Prozesse eine gemeinsame Kapselzugehörigkeit überprüft (daher Bottom-Up). Da die bereits erkannten Kapseln aus dem ersten Schritt (komposite) Prozesse darstellen, werden diese auch als solche behandelt und fließen in die Prüfung ein. Die enthaltenen Subprozesse werden dabei nicht mehr mit behandelt.

4) Es entsteht eine Prozesshierarchie, wobei in Kapseln wiederum Kapseln bzw. Kapseln und/oder elementare Prozesse enthalten sein können. Der Algorithmus endet, wenn alle vorhandenen Prozesse in der gleichen Kapsel sind. Diese Kapsel stellt schließlich den gesamten betrachteten Prozess dar.

Zur Verdeutlichung des angewandten Verfahrens folgt nun ein Beispiel.

Beispiel 4
Gegeben sei der in Abbildung 17 dargestellte Semantikgraph, dessen Kanten bereits semantisch auf eine *StartedAfterFinished*-Relation reduziert wurden. Schritt 1 des Verfahrens wurde daher bereits vollzogen.

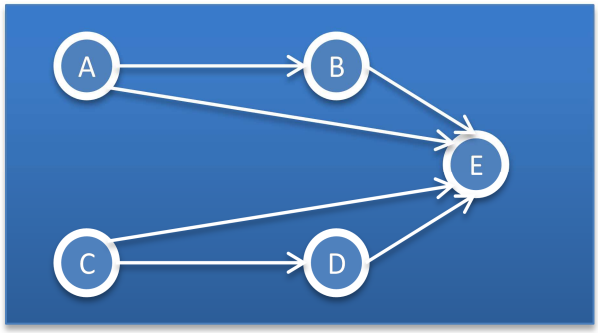

Abbildung 17: Exemplarischer Semantikgraph zur Generierung von Prozesshierarchien

In Schritt 2 des Verfahrens werden nun jeweils zwei Prozesse miteinander verglichen und deren Vorgänger und Nachfolger, soweit vorhanden, analysiert. Betrachtet man beispielsweise die Prozesse A und B so wird ersichtlich, dass sie, abgesehen von der sie verbindenden Kante, den gleichen Nachfolger Prozess E haben. Sie sind daher in der gleichen Kapsel. Diese erhalte exemplarisch die *ID=1*. Werden nun die Prozesse A und C betrachtet zeigt sich, dass A u. a. Prozess B und C u. a. Prozess D als Nachfolger hat, diese Beziehungen jedoch nicht beim jeweils anderen Prozess zu finden sind. Die Prüfung schlägt damit fehl und die Prozesse A und C sind in verschiedenen Kapseln. Die Kapsel von Prozess C erhält die *ID=2*. Führt man das Verfahren mit allen Prozessen fort ergibt sich die in Abbildung 18 links dargestellte erste Kapselung der Prozesse. Die Abbruchbedingung aus Schritt 4 ist nicht erfüllt, weshalb das Verfahren erneut beginnt. Die Kapseln 1 und 2 stellen (komposite) Prozesse dar, weshalb bei diesem Durchlauf nun drei Prozesse (Kapsel 1, Kapsel 2 und der elementare Prozess E) auf eine übergeordnete Kapselung zu

prüfen sind. Da Kapsel 1 und Kapsel 2 keine Vorgänger und nur den gemeinsamen Nachfolger E besitzen, sind sie in einer übergeordneten Kapsel mit *ID=4*. Diese Situation ist in Abbildung 18 rechts dargestellt.

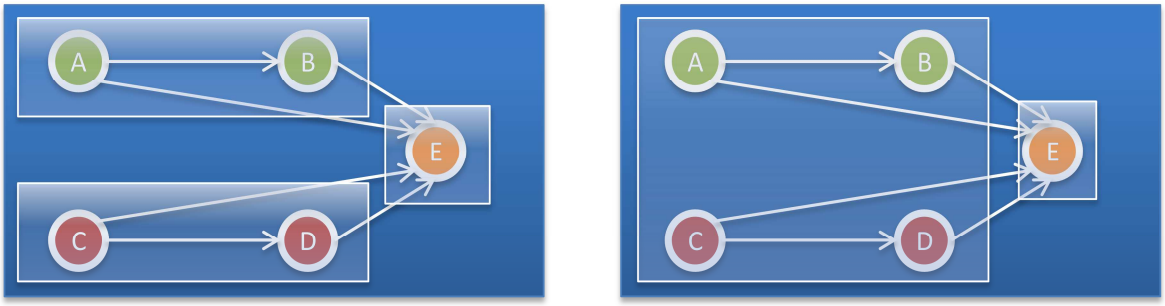

Abbildung 18: Erkennung von Kapseln innerhalb des Semantikgraphs

Nach dem nächsten Durchlauf ergibt sich eine Kapsel in der sämtliche Prozesse enthalten sind, somit wurde die Abbruchbedingung erreicht und der Algorithmus terminiert. Die resultierenden Prozesshierarchien sind in Abbildung 19 dargestellt. Am Ende jeder Hierarchie befindet sich der elementare Prozess aus dem Semantikgraphen.

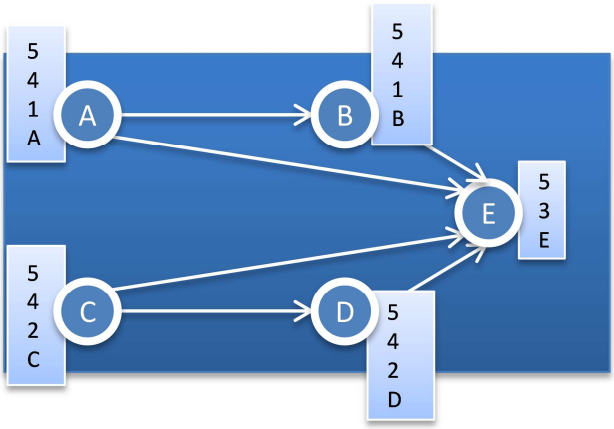

Abbildung 19: Resultierende Prozesshierarchien

Zur besseren Vorstellung wird in Abbildung 20 die Prozesshierarchie exemplarisch anhand eines Prozessmodells dargestellt. Das zu Grunde liegende Meta-Modell stellt elementare Prozesse explizit als Kreise mit Bezeichner dar. Kapseln werden als Boxen um die untergeordneten Prozesse modelliert, wobei deren, durch den Algorithmus generierte, ID innerhalb der Box angezeigt wird. Da sämtliche Prozesse innerhalb der Kapsel 4 beendet werden bevor Prozess E startet, muss nur noch einmal ein entsprechendes Modellierungskonstrukt zwischen der Kapsel 4 und dem Prozess E eingefügt werden.

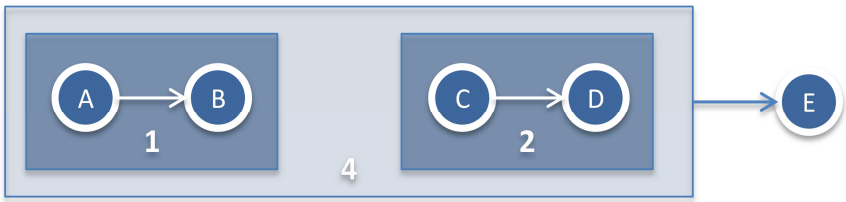

Abbildung 20: Exemplarische Darstellung der kompositen Prozesse als Prozessmodell

Da zwischen Kapsel 1 und Kapsel 2 keine Kontrollflusssemantik entdeckt werden konnte, sind diese im resultierenden Modell nicht verbunden.

4.4 Generierung von Empfehlungen

Im Grunde genommen enthält der Semantikgraph nach der Analyse der funktionalen Perspektive und der Generierung der Prozesshierarchien nun sämtliche Informationen, welche durch die Einbeziehung von Semantikdefinitionen aus dem Log extrahiert werden konnten. Nichtsdestotrotz werden bisher Fehler und Ausnahmen nicht betrachtet. Welche negativen Auswirkungen dies auf Schlussfolgerungen haben kann, zeigt Beispiel 5.

Beispiel 5

Angenommen es existieren 20 Anwendungsfälle in einem Log. In 19 Fällen wird Prozess X vor Prozess Y ausgeführt. Nur in einem Fall bestand eine Ausnahme und Prozess Y wurde vor Prozess X ausgeführt. Das bedeutet, dass in 95% der Fälle eine Nachfolgerbeziehung von X in Richtung Y existiert. Da sich jedoch eine Instanz als Gegenbeispiel finden lässt, würde von der semantischen Interpretation beispielsweise keine *StrictControlFlowConnector*-Semantik entdeckt werden. In einem resultierenden Prozessmodell sollte jedoch trotz der Ausnahme die Information Eingang finden, dass die Ausführungsreihenfolge X → Y bevorzugt wird.

Die Darstellung dieser Information wird über eine *Empfehlung* (*Recommendation*) realisiert und belegt damit ein eigenes vordefiniertes Modellierungskonstrukt. Auch im Fall dessen, dass es sich nicht um eine Ausnahme handelt, können Empfehlungen nützlich sein.

Beispiel 6

Angenommen es bestünden keine Restriktionen bei der Ausführungsreihenfolge zweier Prozesse, d.h. aus Sicht der korrekten Ausführung ist es völlig irrelevant in welcher Reihenfolge zwei Schritte X und Y ausgeführt werden. In diesem Fall kann es ebenso nützlich sein eine Empfehlung angezeigt zu bekommen, welcher der beiden Schritte beispielsweise weniger Zeit benötigt um Engpässen zu entgehen.

Es wird ersichtlich, dass die Anwendungsmöglichkeiten von Empfehlungen innerhalb des Prozessmodells weitreichend sind. Eine detaillierte Einführung in die Notwendigkeit und die Möglichkeiten von Empfehlungen in der Prozessausführung findet sich in [30].

4.4.1 Bestimmung von Zusammenhangskomponenten

Empfehlungen sollen nur dann als Verbindung zweier Prozesse im Prozessmodell dienen sofern nicht bereits ein semantisch definiertes Modellierungskonstrukt der verhaltensorientierten Perspektive diese Aufgabe erfüllt. Der erste Schritt zur Generierung von Empfehlungen ist daher die Suche nach Prozessen, welche noch keine Relation zueinander besitzen. Für diese Kombinationen an Prozessen kann anschließend eine mögliche Generierung einer Empfehlung erfolgen. Dieses Ziel wird wie folgt erreicht:

1) Generierung von Teilgraphen für jede Kapsel
2) Bestimmung der Zusammenhangskomponenten (ZK)

Zur Generierung von Teilgraphen für jede Kapsel werden die Knoten des Semantikgraphen durchlaufen und anhand der gespeicherten Prozesshierarchie einem Kapsel-Graphen zugeteilt. Legt man den Semantikgraph aus Abschnitt 4.3.3 zu Grunde, so ergeben sich die in Abbildung 21 dargestellten Teilgraphen. Im Graphen des Prozesses *Kapsel 4* sind die Prozesse *Kapsel 1* und *Kapsel 2* enthalten. Der Prozess *Kapsel 1* enthält die elementaren Prozesse A und B, wohingegen der Prozess Kapsel 2 die elementaren Prozesse C und D enthält.

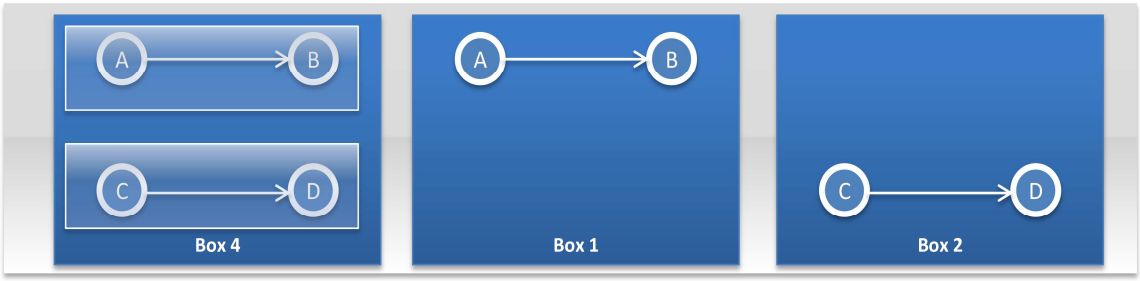

Abbildung 21: Teilgraphen zu jedem kompositen Prozess (Kapsel)

Nun werden für jeden Teilgraph die schwachen Zusammenhangskomponenten bestimmt. Diese Komponenten stehen bisher in keiner Beziehung miteinander und könnten eventuell durch eine Empfehlung verknüpft werden. Der Algorithmus zur Bestimmung der Zusammenhangskomponenten eines Graphen findet sich in [31]. Demnach erfolgt die Bestimmung wie folgt:

```
1) Markiere alle Knoten mit 0
2) Setze c = 0
3) Für alle Knoten v
        wenn v mit 0 markiert ist
              setze c = c + 1
              Tiefensuche(v)
```

Die für dieses Verfahren verwendete Funktion *Tiefensuche(v)* ist folgende:

```
1) markiere den Knoten v mit der Zahl c
2) Für alle Nachbarknoten w von v
        wenn w noch mit 0 markiert ist
                    Tiefensuche(w)
```

Die resultierenden ZKs des Beispiels aus Abbildung 21 ergeben sich anhand des Algorithmus wie in Abbildung 22 dargestellt.

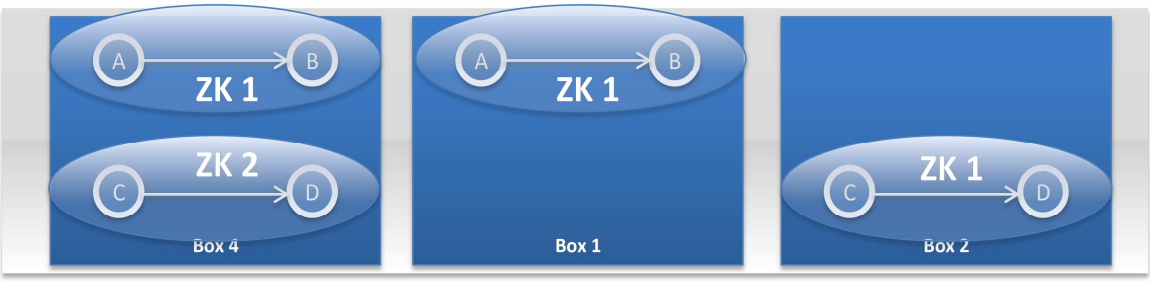

Abbildung 22: Resultierende Zusammenhangskomponenten anhand des Algorithmus von [31]

4.4.2 Überprüfung einer Recommendation-Bedingung

Ist die berechnete Anzahl an ZKs der Box gleich 1, muss diese Kapsel nicht mehr weiter bezgl. einer Empfehlung betrachtet werden. Das Einfügen eines Empfehlungskonstrukts ist hier nicht möglich. Ist hingegen die Anzahl an ZKs größer als 1, so kann für je ein Paar an Elementen aus jeder ZK eine gewünschte, vordefinierte Bedingung (*Recommendation-Bedingung*) überprüft werden. Eine ausführliche Beschreibung möglicher Bedingungen für Empfehlungen findet sich in [30]. Mögliche Bedingungen für eine Empfehlung zwischen den Prozessen X und Y könnten beispielsweise sein:

- Prozess X wird häufiger vor Prozess Y ausgeführt als vice versa
- Prozess X benötigt weniger Zeit als Prozess Y

Bei positiver Prüfung wird ein *Recommendation*-Connector mit spezifischer Richtung in den Semantikgraph eingefügt.

4.5 Zusammenfassung der extrahierten Information

Nach Analyse des Logs liegt nun mit dem Semantikgraph eine allgemeingültige Datenstruktur vor, welche sämtliche im Log auftretende Prozesse als Knoten enthält. Anhand von vordefinierten Semantiken, welche *SemanticPM* als Input dienen, werden den Prozessen Eigenschaften (Attributes) zugewiesen und Relationen eines bestimmten Typs (Edges) zwischen den Prozessen dargestellt. Es sei dabei hervorgehoben, dass die aufzufindenden Semantiken, respektive Regeln, beliebig definiert und flexibel hinzugefügt oder entfernt werden können. Die Relationen und Eigenschaften umfassen alle Perspektiven des POPM. Die funktionale Perspektive (atomare bzw. komposite Prozesse) wird durch die den Knoten hinzugefügten Prozesshierarchien abgedeckt. Wurde zwischen zwei (elementaren oder kompositen) Prozessen keine Kontrollfluss-Semantik entdeckt, konnte eventuell auf Basis einer vordefinierten aber ebenso flexibel auswählbaren Bedingung eine Kontrollfluss-Empfehlung hinzugefügt werden.

- Aufgrund der unterschiedlichen Modellierungsweisen kann *SemanticPM* im Grunde nicht mit den Process Mining Verfahren nach van der Aalst et al. verglichen werden. Die imperativen Prozessmodelle (vgl. Petri-Netze) der Algorithmen aus Kapitel 3 veranschaulichen den Kontrollfluss eines Prozesses sehr detailliert und geben jeden möglichen Weg explizit an.

- Durch die Zuweisung von komplexen Semantiken zu entsprechenden (beliebig definierbaren) Modellierungskonstrukten werden die dem Log zu Grunde liegenden Prozesse teilweise implizit, d.h. deklarativ, modelliert. Anstatt jeden Weg explizit anzugeben, ist diese Information bereits innerhalb eines Konstrukts selbst enthalten.

Abschließend werden die Möglichkeiten des vorgestellten Ansatzes mit den definierten Zielen aus Abschnitt 4.1.1 verglichen.

- SemanticPM generiert eine allgemeingültige Basis-Datenstruktur, welche ein flexibles Process Mining nach beliebig definierten Semantiken ermöglicht.

Dieses Ziel wurde durch die Generierung der Instanzgraphen erreicht. Diese beinhalten alle vorkommenden Prozesse, deren ausführende Agenten, verwendete Daten, verwendete Tools sowie deren Beziehungen zueinander innerhalb einer Instanz. Diese Graphen stellen eine allgemeingültige Datenstruktur dar, die ein Mining nach beliebig vordefinierten Semantiken ermöglicht.

- Es wird eine kombinierte Betrachtung unter Einbeziehung sämtlicher Perspektiven des POPM innerhalb eines Verfahrens angestrebt und damit die Möglichkeit zum Auffinden von Zusammenhängen zwischen den einzelnen Perspektiven gegeben.

Die vordefinierten Semantiken können Bedingungen und Konsequenzen aus allen Perspektiven des POPM enthalten und stellen damit perspektivenübergreifende Zusammenhänge dar. Durch die semantische Interpretation und der algorithmischen Suche nach Regeln werden die Instanzgraphen perspektivenübergreifend analysiert und die bewiesenen Zusammenhänge im resultierenden Semantikgraph festgehalten.

- SemanticPM strebt eine exakte Beschreibung des protokollierten Verhaltens im Process Execution Logs nach der Closed-World Annahme unter Einbeziehung von Fehlern und Ausnahmen an.

Die Instanzgraphen stellen ein exaktes Abbild der einzelnen Anwendungsfälle dar, welche im Log auftreten. Enthält die Aufzeichnung eine Ausnahme oder einen Fehler, so ist dies auch innerhalb der Instanzgraphen nachzuvollziehen. Wird eine vorhandene Semantik aufgrund einer Ausnahme oder eines Fehlers durch die semantische Interpretation nicht erkannt, wird mit Hilfe der im letzten Schritt generierten Empfehlungen die Situation dennoch offensichtlich dargestellt.

- Ziel ist eine optimale Beschreibung des Process Execution Logs im Sinne eines minimalen aber vollständig beschriebenen Ergebnisraumes durch ein resultierendes Prozessmodell.

Die Analyse der Instanzgraphen generiert den Semantikgraphen, welcher den Ergebnisraum vollständig und minimal beschreibt. Der Semantikgraph lässt sich nur in Modelle überführen, welche exakt die gleichen Ausführungsreihenfolgen ermöglichen, wie sie im Log aufgezeichnet wurden. Dies wird auch durch die Anwesenheit von Ausnahmen nicht verändert, da evtl. generierte Empfehlungskonstrukte keine echten Constraints im Sinne der Ausführung darstellen, sondern lediglich visualisierte Empfehlungen für die Anwender sind.

Somit wurden alle definierten Ziele durch den Process Mining Ansatz SemanticPM erreicht. In Kapitel 5 wird *SemanticPM* exemplarisch auf Basis ausgewählter Semantikdefinitionen implementiert und dessen Funktionsfähigkeit anhand eines (beispielhaften) Logs überprüft.

5 Exemplarische Implementierung auf Basis ausgewählter Semantikdefinitionen

In diesem Kapitel wird eine exemplarische Implementierung des im vorherigen Kapitel vorgestellten Process Mining Ansatzes *SemanticPM* erläutert. Diese Beschreibung dient außerdem als Anleitung zur Implementierung der Suche nach beliebigen Regeln innerhalb des Logs.

5.1 Grundlegende Analyse des Logs

Die Grundlage für die Analyse ist das Einlesen des Logs. Dafür muss das Log geparst und entsprechende Datenstrukturen befüllt werden. In dieser Arbeit wird exemplarisch ein als JSON Datei vorliegendes Log aus dem Workflow Management System (WfMS) Process Navigator mit der Process Execution Engine EsproNa [32] verwendet.

5.1.1 Parsen der JSON Logdatei

Wie das theoretische Modell aus Kapitel 4 besteht auch das Log des WfMS aus einer Vielzahl an Ereignissen, welche jeweils ein 8-Tupel an Ereignisinformation darstellen. Einen Ausschnitt aus einer JSON-Logdatei zeigt folgendes Listing:

```
{
        Event: 1,
        Time: "1900-01-01 00:00:00",
        Action: "start",
        Case: 1,
        Process: "A",
        Agent: ["http://ai4.inf.uni-bayreuth.de/ontology/individuals#Jack"],
        Data: [],
        Tool: ["HIS"]
},
{
        Event: 2,
        Time: "1900-01-02 00:00:00",
        Action: "finish",
        Case: 1,
        Process: "A",
        Agent: ["http://ai4.inf.uni-bayreuth.de/ontology/individuals#Jack"],
        Data: [],
        Tool: ["HIF"]
},
{
        Event: 3,
        Time: "1900-01-03 00:00:00",
        Action: "start",
        Case: 1,
        Process: "B",
        Agent: ["http://ai4.inf.uni-bayreuth.de/ontology/individuals#Jack"],
        Data: [],
        Tool: ["HIF"]
}
```

Diese drei Ereignisse sind alle dem Anwendungsfall *Case 1* des gesamten aufgezeichneten Prozesses zuzuordnen. Die einzelnen Zeilen stellen wie in Tabelle 8 bereits theoretisch erläutert die aufgezeichneten Prozessdaten dar. Die protokollierten Agenten, Daten und Tools enthalten ein Array an Elementen pro Ereignis. Zum Parsen dieses Event-Logs wurde das Google GSON Projekt verwendet [33]. Anhand der

Ereignisinformationen wird eine generische Liste aufgebaut, welche Objekte der Klasse *Event* enthält. Diese Klasse stellt die unterste Ebene der Process Mining Prozedur dar und enthält folgende Felder:

```
int eventID;
Process processID;
int caseID;
String action;
List<Agent> agents;
List<Data> data;
List<Tool> tools;
Date timestamp;
```

Anhand der vom JSON Parser gelieferten Daten werden temporäre Variablen entsprechenden Typs befüllt und der Liste schrittweise die Ereignisse des Logs hinzugefügt:

```
eventID = gson.fromJson(eventObject.get("Event"), int.class);
datetime = gson.fromJson(eventObject.get("Time"), String.class);
timestamp = df.parse(datetime);
action = gson.fromJson(eventObject.get("Action"), String.class);
caseID = gson.fromJson(eventObject.get("Case"), int.class);
pid = gson.fromJson(eventObject.get("Process"), String.class);
process = new Process(pid);

agentArray = eventObject.get("Agent").getAsJsonArray();
for(int j = 0; j < agentArray.size(); j++)
       agents.add(new Agent(gson.fromJson(agentArray.get(j), String.class)));

dataArray = eventObject.get("Data").getAsJsonArray();
for(int j = 0; j < agentArray.size(); j++)
       data.add(new Data(gson.fromJson(dataArray.get(j), String.class)));

toolArray = eventObject.get("Tool").getAsJsonArray();
for(int j = 0; j < agentArray.size(); j++)
       tools.add(new Tool(gson.fromJson(toolArray.get(j), String.class)));

events.add(new Event(eventID, timestamp, action, caseID, process, agents, data,
tools));
```

Das Ereignis-Log wurde somit eingelesen und in eine Liste von Event-Objekten geschrieben.

5.1.2 Aufbau instanzspezifischer Eventlisten

Der nächste Schritt ist die Aufspaltung der Event-Liste nach den verschiedenen Anwendungsfällen/Instanzen, welche innerhalb des Logs auftreten. Hierfür wird die Event-Liste einmal durchlaufen und die Event Objekte anhand deren Case ID einer separaten Event-Liste pro Anwendungsfall zugeordnet. Die Klasse *Case* hat dabei folgende Felder:

```
int caseID;
List<Event> events;
```

Nach Aufteilung der Ereignisse ist eine Liste an verschiedenen Anwendungsfällen vorhanden, welche jeweils die zugeordneten Ereignisse enthält.

```
List<Case> cases = new ArrayList<Case>();
```

Des Weiteren wird schrittweise eine Liste für alle auftretenden Prozesse aufgebaut. Diese wird später benötigt um auf einfache Weise zu überprüfen ob zwischen einer dualen Kombination an Prozessen eine bestimmte Beziehung besteht.

```
List<Process> processes;

if(!processes.contains(event.getProcessID()))
      processes.add(event.getProcessID());
```

Da nun die Ereignisse anhand der CaseID einzelnen Instanzen zugeordnet wurden, stehen nun genügt Informationen zur Verfügung um Instanzgraphen zu erzeugen.

5.2 Erzeugung und Implementierung der Instanzgraphen

Als grundlegende Graph-Datenstruktur wird in dieser Arbeit JGraphT [34] verwendet. Zur Definition des Instanzgraphen wurden die Default-Implementierungen der Knoten und Kanten durch entsprechende eigene Definitionen ersetzt. Ein Knoten innerhalb eines Instanzgraphen besitzt folgende Struktur:

```
class InstanceProcessNode

        Process processID;
        List<Agent> agents;
        List<Data> data;
        List<Tool> tools;
```

Eine Kante des Instanzgraphen hat folgenden Aufbau:

```
class InstanceEdge<V> extends DefaultEdge

        V v1;
        V v2;
        String transitivity;
        String type;
```

Eine Kante hat somit Referenzen auf die beiden Knoten, die durch sie verbunden werden, ein Feld für den Typ der Kante, d.h. handelt es sich um eine Nachfolger oder einen parallelen Prozess, sowie ein Feld für die Transitivität, d.h. ist es eine direkte oder eine transitive Beziehung. Vor jeder Analyse einer Instanz wird ein neuer (leerer) Graph erzeugt. Es wird ein gerichteter Graph verwendet, da die dargestellten Beziehungen eine zeitliche Abfolge beschreiben:

```
currentGraph = new DirectedGraph<InstanceProcessNode, InstanceEdge>(
          new ClassBasedEdgeFactory<InstanceProcessNode,
          InstanceEdge>(InstanceEdge.class));
```

Da im Allgemeinen mehrere Instanzen innerhalb eines Logs auftreten, werden die einzelnen Instanzgraphen in eine Liste einsortiert. Diese kann später durch die semantische Interpretation durchlaufen werden.

```
List<DirectedGraph> instanceGraphs = new ArrayList<DirectedGraph>();
```

Zur Erzeugung eines Instanzgraphen, wird die Event-Liste eines Anwendungsfalls durchlaufen und jeweils zwei zeitlich aufeinanderfolgende Ereignisse miteinander verglichen. Ist im Instanzgraphen für den betrachteten Prozess noch kein Knoten vorhanden, so wird dieser zuerst erzeugt:

```
node = new InstanceProcessNode(event.getProcessID(), event.getAgents(),
        event.getData(), event.getTools());

if(newVertex)
        currentGraph.addVertex(node);
```

Es folgt nun ein Listing, welches die Klassifikation einer dualen Event Beziehung vornimmt und dementsprechend eine neue Kante innerhalb des Graphen generiert.

```
if(event1.getAction().equals("start") && event2.getAction().equals("start"))
        currentGraph.addEdge(node1, node2,
        new InstanceEdge<InstanceProcessNode>(node1, node2, direct, parallel));

else if(event1.getAction().equals("start") && event2.getAction().equals("finish"))
```

```
            currentGraph.addEdge(node1, node2,
            new InstanceEdge<InstanceProcessNode>(node1, node2, direct, parallel));

        else if(event1.getAction().equals("finish") && event2.getAction().equals("finish"))
            currentGraph.addEdge(node1, node2,
            new InstanceEdge<InstanceProcessNode>(node1, node2, direct, parallel));

        else if(event1.getAction().equals("finish") && event2.getAction().equals("start"))
            currentGraph.addEdge(node1, node2,
            new InstanceEdge<InstanceProcessNode>(node1, node2, direct, successor));
```

Hierdurch wird die Klassifikation, wie in Abschnitt 4.2.1 beschrieben, durchgeführt und bei entsprechender Bedingung eine neue Kante innerhalb des gerade betrachteten Instanzgraphen erzeugt. Bei diesen Kanten handelt es sich jedoch bisher nur um direkte Kanten. Transitive Kanten werden erst in einem nachgelagerten Schritt hinzugefügt, welcher den transitiven Abschluss zwischen den Knoten und deren Kanten herstellt. Hierfür wurde beim Durchlaufen der Ereignisse die Reihenfolge der Prozesse (Prozesskette, Trace) festgehalten:

```
currentTrace = new ArrayList<InstanceProcessNode>();

if(!currentTrace.contains(node))
        currentTrace.add(node);
```

Anhand der gespeicherten Prozesskette kann nun der transitive Abschluss gebildet und in den Graphen eingefügt werden. Dieser wird durch die folgende Prozedur realisiert:

```
void transitiveClosure(DirectedGraph currentGraph, List<InstanceProcessNode> trace)
{
        Object[] traceArray = trace.toArray();

        for(int i = 0; i < traceArray.length; i++)
        {
                node1 = (InstanceProcessNode) traceArray[i];
                for(int j = i+1; j < traceArray.length; j++)
                {
                        node2 = (InstanceProcessNode) traceArray[j];
                        currentGraph.addEdge(node1, node2,
                                new InstanceEdge< InstanceProcessNode>(node1, node2,
                                transitiv, successor));
                }
        }
}
```

Nachdem auf diese Weise alle Event-Listen der einzelnen Anwendungsfälle durchlaufen wurden, stehen alle Instanzgraphen in der Liste `instanceGraphs` zur Verfügung. Diese Liste dient nun der semantischen Interpretation als Input.

5.3 Implementierung der semantischen Interpretation

5.3.1 Grundlegende Vorgehensweise und Architektur

Wie in Kapitel 4 beschrieben, lassen sich Semantiken in Relationen zwischen Prozessen und Eigenschaften einzelner Prozesse aufteilen, je nach Wertigkeit der Regel. Diese werden daher in zwei verschiedenen generischen Listen abgelegt, welche einerseits Objekte vom Typ `IEdgeType` und andererseits Objekte vom Typ `IAttributeType` enthalten.

```
List<IEdgeType> connectors;
List<IAttributeType> attributes;
```

Diese beiden Typen stellen Interfaces dar, die von konkreten Klassen, welche Semantiken repräsentieren, implementiert werden. Beide Interfaces definieren die Methode `getType()`, welche den Typ der Semantik zurückliefert. Des Weiteren liefert die Methode `constraintClass()` eine Zahl zurück, nach der die einzelnen Semantiken in eine Abarbeitungsreihenfolge gebracht werden können, so dass allgemeinere Regeln zuerst überprüft werden. Der Unterschied zwischen den beiden Typen zeigt sich in der Methode `proof`. Beim `IEdgeType` werden zwei Prozesse miteinander in Beziehung gesetzt, wohingegen beim `IAttributeType` nur die Eigenschaften eines Prozesses überprüft werden. Die `isControlFlow`-Methode gibt an, ob eine Edge der verhaltensorientieren Perspektive zuzuordnen ist oder nicht. Diese Information wird später benötigt um den Semantikgraph zur Generierung der Kapselung entsprechend auf Kanten dieses Typs zu reduzieren. In Abbildung 23 ist ein Klassendiagramm dargestellt, welches die exemplarische Implementierung der beiden Interfaces durch zwei Connector-Klassen und durch eine Attribute Klasse beschreibt.

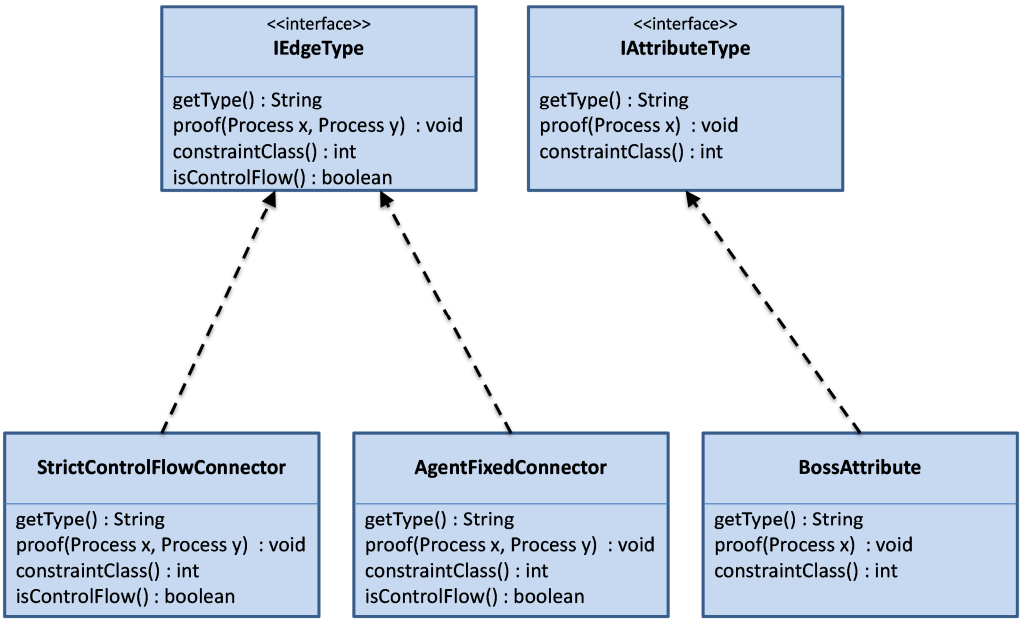

Abbildung 23: Exemplarisches Klassendiagramm der semantischen Interpretation

Jede vorhandene Klasse, die eine Semantik repräsentiert, wird in der Funktion `loadRules()` instanziiert und der entsprechenden `connectors`- oder `attributes`-Liste zugeordnet. Jede neu definierte Regel welche überprüft werden soll, muss folglich innerhalb der `loadRules()` einer dieser Listen hinzugefügt werden. Anschließend werden die Listen anhand deren ConstraintClass geordnet und so eine Abarbeitungsreihenfolge hergestellt. Nach dem sämtliche Regeln geladen wurden, kann die semantische Interpretation ausgeführt werden:

```
for(IEdgeType connector : connectors)
	for(Process x : processes)
		for(Process y : processes)
			if(!x.getPid().equals(y.getPid()))
				connector.proof(x, y, successorGraphs, semanticGraph);

for(IAttributeType attribute : attributes)
	for(Process x : processes)
		attribute.proof(x, successorGraphs, semanticGraph);
```

Der erste Absatz beschreibt dabei die Überprüfung der zweiwertigen Regeln (Edges). Für jede Kombination aus Prozessen x und y aus der Liste aller vorhandenen Prozesse wird die aktuell betrachtete Semantik überprüft. Hierfür wird die Methode `proof` aufgerufen, deren Implementierung im folgenden Abschnitt detailliert erläutert wird.

5.3.2 Exemplarische Implementierungen des Interfaces IEdgeType

Da die Implementierung des Interfaces IEdgeType grundlegend für die Erweiterung der semantischen Interpretation mit beliebigen Semantiken ist, wird hier exemplarisch die Realisierung von zwei Semantikdefinitionen aufgezeigt. Eine erste exemplarische Implementierung des Interfaces IEdgeType ist die Klasse *StrictControlFlowConnector*. Zur besseren Übersicht folgt noch einmal die zu Grunde liegende Regel in SWRL-Notation:

```
StrictControlFlowConnector(?SC) & From(?SC, ?P1) & To(?SC, ?P2)  ⇨
                    startedAfterFinished(?P2, ?P1)
```

Anhand dieser Regel wird nun die Funktion `proof` erläutert. Wie in Kapitel 4 beschrieben, müssen die Klassenzugehörigkeiten *From* und *To* nicht mehr überprüft werden, da dies bereits durch die Belegung der Parameter, welche der Funktion übergeben werden, geschehen ist. Es folgt nun ein Listing, welches die wichtigsten Elemente der Funktion zeigt. Die Erläuterung stellt gleichzeitig eine Anleitung zur Implementierung beliebiger Repräsentation von Semantikdefinitionen dar, nach denen im Log gesucht werden soll.

```
1       void proof(Process x, Process y, List<DirectedGraph> instanceGraphs, DirectedGraph
2               semanticGraph)
3       {
4               boolean proof = true;
5
6               for(DirectedGraph graph : instanceGraphs)
7               {
8                       nodeX = findInstanceNode(x, graph);
9                       nodeY = findInstanceNode(y, graph);
10
11                      edge = findSuccessorEdge(nodeX, nodeY, graph);
12
13                      if(edge.getTransitivity().equals("direct"))
14                          transitivity = "direct";
15
16                      if(null == edge || !edge.getType().equals("successor"))
17                          proof = false;
18              }
19
20              if(proof)
21              {
22                      semanticNodeX = findSemanticNode(x, semanticGraph);
23                      semanticNodeY = findSemanticNode(y, semanticGraph);
24
25                      if(null == semanticNodeX)
26                              semanticGraph.addVertex(semanticNodeX);
27
28                      if(null == semanticNodeY)
29                              semanticGraph.addVertex(semanticNodeY);
30
31                      if(transitivity.equals("transitiv"))
32                              semanticGraph.addEdge(semanticNodeX, semanticNodeY, new
33                              SemanticEdge(semanticNodeX, semanticNodeY, "transitiv", this));
34
35                      else
36                              semanticGraph.addEdge(semanticNodeX, semanticNodeY, new
37                              SemanticEdge(semanticNodeX, semanticNodeY, "direct", this));
38              }
39      }
```

Die Parameter der Funktion sind die zwei zu vergleichenden Prozesse x und y, die Liste an Instanzgraphen sowie der Semantikgraph, in welchen potentiell erkannte Semantiken eingefügt werden sollen. In Zeile 4 wird die Beweisvariable `proof` auf `true` gesetzt, da ein Widerspruchsbeweis angewendet wird. In den Zeilen 6 bis 21 erfolgt nun der Widerspruchsbeweis. Für jeden Instanzengraphen der übergebenen Liste

werden zuerst die jeweiligen Knoten der Prozesse gesucht (Zeile 8 und 9). Das geschieht über eine entsprechende Hilfsmethode `findInstanceNode(Process x, DirectedGraph instanceGraph)`. Da die *StrictControlFlowConnector*-Regel keine Bedingungen auf der linken Seite enthält, entfällt eine umgebende IF-Bedingung in der etwaige Bedingungen überprüft werden würden. Die rechte Seite der Regel ist die Property *StartedAfterFinished*, d.h. es ist zu prüfen, ob Prozess y stets nach dem Ende von Prozess x gestartet wurde. Dementsprechend liefert die Funktion `findSuccessorEdge` in Zeile 11 die Kante des aktuell betrachteten Instanzgraphen, welcher die beiden Prozesse potentiell verbindet. In Zeile 13 bis 14 wird bei Betrachtung einer direkten Kante auch der Typ der in den Semantikgraph einfließenden Kante auf direkt gesetzt. Direkte Kanten müssen bei der Überführung in ein Prozessmodell betrachtet werden. Zeile 16 und 17 (rot markiert) stellen nun die Überprüfung der rechten Seite und somit der *StartedAfterFinished*-Property dar. Wenn keine Kante durch `findSuccessorEdge` zurückgeliefert oder wenn eine Kante existiert aber der Typ nicht *successor* ist, dann wurde keine Nachfolgerbeziehung nachgewiesen und somit ein Gegenbeispiel gefunden. Die Annahme wurde widerlegt und `proof` wird auf `false` gesetzt. Wurde jedoch bewiesen, dass die aktuelle Semantik für die betrachteten Prozesse gültig ist, werden in Block 20 bis 39 entsprechende Knoten und eine Kante in den Semantikgraphen eingefügt. Da den Prozess repräsentierende Knoten bereits durch proof-Methoden anderer Klassen angelegt worden sein könnten, wird durch die Hilfsmethode `findSemanticNode` nach einem bereits bestehenden Knoten dieses Prozesses gesucht. Wurde kein bestehender Knoten gefunden, kann in Zeile 26 ein neuer Knoten angelegt und dem Semantikgraphen hinzugefügt werden. In den Zeilen 31 bis 37 wird schließlich eine Kante im Semantikgraph angelegt. Diese hat entweder den Typ `transitiv` oder `direct` und repräsentiert durch den Parameter `this` die aktuell geprüfte Semantik, in diesem Fall den *IEdgeType StrictControlFlowConnector*.

Eine zweite Implementierung des Interfaces IEdgeType ist die Klasse *BossConnector,* welche die Semantik des laufenden Beispiels aus Kapitel 4 aufzufinden versucht. Die Regel lautet in SWRL-Notation:

```
BossConnector(?BC) & From(?BC, ?P1) & To(?BC, ?P2) & hasAgent(?P1, ?A) &
NotBoss(?A) & hasAgent(?P2, ?B) & NotBoss(?B)  ⇨  startedAfterFinished(?P2, ?P1)
```

Die `proof`-Funktion ist nahezu identisch zu der des *StrictControlFlowConnectors*, lediglich die angesprochene umgebende IF-Bedingung zur Überprüfung der Bedingungen der linken Seite der Regel muss hinzugefügt werden. Für die for-Schleife, welche die Instanzgraphen durchläuft ergibt sich folglich für den *BossConnector* folgende Implementierung:

```
for(DirectedGraph graph : instanceGraphs)
{
        nodeX = findInstanceNode(x, graph);
        nodeY = findInstanceNode(y, graph);

        edge = findSuccessorEdge(nodeX, nodeY, graph);

        if(edge.getTransitivity().equals("direct"))
                transitivity = "direct";

        if(notBoss(nodeX.getAgents()) && notBoss(nodeY.getAgents()))
        {
                if(null == edge || !edge.getType().equals("successor"))
                        proof = false;
        }
}
```

Die rot markierte Zeile stellt die Überprüfung der Bedingungen der Regel dar. Hier erfolgen zwei Aufrufe einer Hilfsmethode `notBoss(List<Agent>)`. Diese stellt eine Verbindung zu einer Ontologie her und extrahiert daraus anhand der übergebenen Agent Identifier ob es sich um einen Boss oder nicht handelt.

Diese Vorgehensweise kann nun als Anleitung zur flexiblen Umsetzung für sämtliche Semantikdefinition genutzt werden, welche der verhaltensorientierten, organisationsorientierten, datenorientierten oder operationsorientieren Perspektive zuzuordnen sind.

5.3.3 Implementierung der Prozesskapselung

Der letzte Schritt der semantischen Interpretation ist die Kapselung von Prozessen, d.h. die Generierung von Prozesshierarchien. Wie in Kapitel 4 beschrieben, beruht die funktionale Perspektive auf der verhaltensorientierten Perspektive und ist daher erst nach der Generierung des Semantikgraphen zu behandeln. Daher folgt nach der Abarbeitung aller anderen Semantikdefinitionen der Aufruf

```
semanticGraph = generateProcessHierarchies(semanticGraph, processes);
```

Die noch leeren Prozesshierarchien des Semantikgraphen werden durch die Methode `generateProcessHierarchies` generiert und der Semantikgraph mit generierten Prozesshierarchien wieder in die gleiche Variable geschrieben. Die Kapselung der Prozesse des Semantikgraphen verläuft in der Implementierung in zwei Phasen:

1) Initiale Kapselung von rein elementaren Prozessen
2) Kapselung von bereits gekapselten und elementaren Prozessen

Die unterschiedliche Handhabung der Kapselung bei elementaren und kompositen Prozessen ist deshalb notwendig, da bei elementaren Prozessen keine internen Kanten existieren, welche die Überprüfung auf Kantengleichheit verhindern und deshalb entfernt werden müssen. Hierzu aber später mehr. Beim ersten Durchlauf der Prozesse werden die aktuell betrachteten Prozessknoten durch eine Hilfsmethode `findSemanticNode` im Semantikgraphen aufgefunden:

```
nodeX = findSemanticNode(processX, semanticGraphCaps);
nodeY = findSemanticNode(processY, semanticGraphCaps);
```

Anschließend werden wiederum über zwei Hilfsmethoden `getIncomingEdges` und `getOutgoingEdges` die jeweiligen Kanten der beiden Knoten aus dem Semantikgraphen extrahiert:

```
List<SemanticEdge> incomingEdgesOfX = getIncomingEdges(nodeX, semanticGraph);
List<SemanticEdge> outgoingEdgesOfX = getOutgoingEdges(nodeX, semanticGraph);
```

Die Überprüfung ob zwei elementare in der gleichen Kapsel sind, wird nun wie folgt realisiert: Für jede eingehende Kante des Knotens X mit Start bei Knoten V1 muss auch eine eingehende Kante des Knotens Y mit Start bei Knoten V1 vorhanden sein.

```
for(SemanticEdge incomingEdgeOfX : incomingEdgesOfX)
{
	boolean found = false;
	for(SemanticEdge incomingEdgeOfY : incomingEdgesOfY)
	{
		if(incomingEdgeOfX.getV1().equals(incomingEdgeOfY.getV1()))
			found = true;
	}

	if(!found)
		return false;
}
```

Der gleiche Test wird für die ausgehenden Kanten durchgeführt. Wird keine passende Kante innerhalb der `incomingEdgesOfY`-Liste oder der `outgoingEdgesOfY`-Liste gefunden, so wurde die Variable `found` nicht auf

`true` gesetzt und der Algorithmus liefert `false` zurück. Dieser Beweis findet in der `proof`-Methode eines Objektes `connectorCapsule` statt.

```
equiCapsule = connectorCapsule.proof(nodeX, nodeY, semanticGraphCaps);
```

Es liegt eine Verwaltung der gesamten entstehenden Kapseln zu Grunde. Aufgrund der injektiven Prozessabbildung definiert sich eine Kapsel lediglich auf Basis dessen, welche Prozesse sie enthält. Eine eindeutige Beschreibung einer Kapsel liegt somit bereits durch die Angabe der enthaltenen elementaren Prozesse vor. Wurde beispielsweise erkannt, dass `nodeX` und `nodeY` in einer gleichen Kapsel sind, so werden beide einer Liste `includedProcesses` hinzugefügt auf Basis derer eine neue Kapsel instanziiert wird. Dies zeigt folgendes Listing:

```
List<SemanticProcessNode> includedProcesses = new ArrayList<SemanticProcessNode>();
includedProcesses.add(nodeX);
includedProcesses.add(nodeY);

Capsule currentCapsule = new Capsule(capsuleID++, includedProcesses);
nodeX.setCurrentCapsule(currentCapsule);
nodeY.setCurrentCapsule(currentCapsule);
```

Die aktuell erkannten Kapseln werden am Ende jedes Durchlaufs der Hierarchie eines jeden Knotens des Semantikgraphen hinzugefügt:

```
node.getProcessHierarchy(getHierarchy().add(node.getCurrentCapsule());
```

Nach dieser ersten Analyse und Abarbeitung aller elementaren Prozesse sind potentiell Kapseln zu einer Worklist hinzugefügt worden. Diese werden nun in den folgenden Schritten miteinander verglichen und auf eine mögliche weitere Kapselung überprüft bis die Anzahl der Elemente in der Worklist gleich eins ist, d.h. bis der Algorithmus auf der höchsten Ebene angekommen ist und nur noch die Kapsel welche den Gesamtprozess darstellt übrig ist:

```
while(capsuleWorkList.size() > 1)
```

Wie bereits angedeutet müssen Kapseln geringfügig anders behandelt werden wie elementare Prozesse. Um zwei verschiedene Kapseln auf eine weitere Kapselung zu testen, müssen nur ein beliebiger elementarer Prozess der einen Kapsel und ein beliebiger elementarer Prozess der anderen Kapsel verglichen werden. Dies liegt nahe, da für alle Angehörigen einer Kapsel die gleichen Eigenschaften bezgl. der verhaltensorientierten Perspektive gelten. Da eine Kapsel mindestens einen Knoten enthält, wird für das weitere Vorgehen immer das erste Element der enthaltenen Prozesse verwendet:

```
SemanticProcessNode nodeX = x.getIncludedProcesses().get(0);
```

Der obige Knoten `nodeX` besitzt zu diesem Zeitpunkt jedoch noch Kanten zu seinen Nachbarn innerhalb seiner aktuellen Kapsel und Kanten welche in die andere Kapsel münden bzw. aus der anderen Kapsel in ihn münden. Diese müssen zuvor entfernt werden, so dass nur Kanten geprüft werden, welche weder innerhalb noch zwischen den beiden Kapseln verlaufen. Exemplarisch wird dies anhand der eingehenden Kanten des Knotens X gezeigt:

```
List<SemanticEdge> edgesToRemove = new ArrayList<SemanticEdge>();

for(SemanticEdge incomingEdgeOfX : incomingEdgesOfX)
{
       for(SemanticProcessNode node : y.getIncludedProcesses())
              if(incomingEdgeOfX.getV1().equals(node))
```

```
                        edgesToRemove.add(incomingEdgeOfX);

        for(SemanticProcessNode node : x.getIncludedProcesses())
                if(incomingEdgeOfX.getV1().equals(node))
                        edgesToRemove.add(incomingEdgeOfX);
}

for(SemanticEdge edge : edgesToRemove)
        incomingEdgesOfX.remove(edge);
```

Die erste verschachtelte for-Schleife erkennt Kanten, welche aus der anderen Kapsel y in den Knoten X der Kapsel x laufen. Die zweite verschachtelte for-Schleife erkennt Kanten, welche innerhalb der eigenen Kapsel x zum Knoten X zeigen. Die aufgezeichneten Kanten werden schließlich entfernt. Nun kann wieder der gleiche Beweis wie im elementaren Fall erfolgen.

5.4 Implementierung von Empfehlungen

Die Grundlage zur Implementierung von Empfehlungen sind Teilgraphen für jede Kapsel. Diese Graphen werden anhand der enthaltenen elementaren Prozesse jeder Kapsel erzeugt. Es wird zuerst für jede Kapsel ein leerer Graph instanziiert. Anschließend werden die enthaltenen Prozesse durchlaufen und entsprechende Knoten erzeugt. Zuletzt werden Kanten aus dem Semantikgraph übernommen, welche zwischen den erzeugten Knoten verlaufen und der erzeugte Graph der Kapsel zugeteilt:

```
capsule.setCapsulePartGraph(capsuleGraph);
```

Anschließend werden für jede Kapsel deren Zusammenhangskomponenten bestimmt.

```
countConnectedComponents = connectedComponents(capsuleGraph);
```

Die Funktion `connectedComponents(capsuleGraph)` nimmt die Zuteilung der ZK unter den Knoten vor und liefert die Gesamtanzahl an ZKs zurück. Es handelt sich dabei um eine Implementierung des Algorithmus von [31]. Von jedem noch nicht besuchten Knoten aus wird eine Tiefensuche gestartet, welche zu erreichende Knoten mit einer Marke versieht. Das folgende Listing zeigt, wie die Marke c bei Erreichen eines noch nicht besuchten Knotens inkrementiert wird:

```
while(capsuleGraph.vertexSet.hasNext())
{
        node = (SemanticProcessNode) capsuleGraph.vertexSet.next();
        if(node.getConnectedComponent() == 0)
        {
                c++;
                depthFirstSearch(c, node, capsuleGraph);
        }
}

return c;
```

Das folgende Listing zeigt einen Ausschnitt der Tiefensuche. Sämtliche Knoten, die durch die Tiefensuche erreicht werden können, werden mit der Marke c markiert. Der rekursive Aufruf setzt die Suche in Richtung noch nicht besuchter Knoten fort.

```
void depthFirstSearch(int c, SemanticProcessNode node, DirectedGraph capsuleGraph)
{
        node.setConnectedComponent(c);

        while(capsuleGraph.edgeSet.hasNext())
        {
```

```
                edge = (SemanticEdge) capsuleGraph.edgeSet.next();
                node1 = (SemanticProcessNode) edge.getV1();
                node2 = (SemanticProcessNode) edge.getV2();

                if(node1.getConnectedComponent() == 0)
                        depthFirstSearch(c, node1, capsuleGraph);

                if(node2.getConnectedComponent() == 0)
                        depthFirstSearch(c, node2, capsuleGraph);
        }
}
```

Nach Bestimmung der Zusammenhangskomponenten kann nun eine beliebige, vordefinierte Recommendation-Bedingung zwischen Elementen aus den einzelnen ZKs überprüft werden. Auch die gewählte Recommendation-Klasse kann gewählt oder abgewählt werden und ermöglicht dadurch eine flexible Auswahl der zu generierenden Empfehlung. Hierzu wird jeweils ein Knoten aus jeder ZK ausgewählt:

```
occurrenceRecommendation.recommendationRequirement(nodesOfConnectedComponents,
        instanceGraphs, semanticGraphWithRecommendation);
```

Wie bereits in Kapitel 4 erläutert, können die Bedingungen für das Einfügen einer Empfehlung sehr verschieden sein. In diesem Beispiel wird eine Empfehlung zwischen zwei Prozesse x und y hinzugefügt, wenn die Reihenfolge x → y häufiger auftritt als y → x. Innerhalb der Methode `recommendationRequirement` der Klasse `occurrenceRecommendation` wird daher das Auftreten dieser Prozesskombination gezählt, indem die Instanzgraphen durchlaufen werden:

```
int count = 0;
for(DirectedGraph instanceGraph : instanceGraphs)
        if(instanceGraph.containsEdge(nodeX, nodeY))
                count++;
```

Die Häufigkeiten werden schließlich in eine Matrix `frequency[][]` eingetragen. Anhand eines Vergleichs der Elemente der Matrix wird eine neue Kante in den Semantikgraph mit entsprechender Richtung eingefügt:

```
if(frequency[i][j] >= frequency[j][i])
{
        semanticGraph.addEdge(nodeI, nodeJ, new SemanticEdge(nodeI, nodeJ, "direct",
                this));
}
```

Somit wurde die exemplarische Implementierung von Process Mining auf Basis ausgewählter expliziter Semantikdefinitionen dargestellt.

5.5 Komplexitätsbetrachtung

Nachdem nun eine exemplarische Implementierung des Ansatzes erläutert wurde, erfolgt in diesem Abschnitt eine kurze Komplexitätsbetrachtung der Implementierung. Hier bezeichnet m die Anzahl der Anwendungsfälle (Instanzen) innerhalb des Logs und n die maximale Anzahl der Prozessschritte innerhalb einer Instanz. Die kombinierte Betrachtung sämtlicher Perspektiven erfordert einen größeren Aufwand als die isolierte Sichtweise, welche auf eine Perspektive bezogen ist. Bisherige Verfahren senken deren Komplexität vor allem dadurch, dass nach dem ersten Durchlauf des Logs auf einer akkumulierten Datenstruktur gearbeitet werden kann. Ein Punkt, welcher die Komplexität dieses Verfahrens im Gegensatz zu bisherigen Verfahren deshalb enorm erhöht ist die Tatsache, dass für jede Instanz ein eigener Graph aufgebaut und später durchsucht werden muss. Das ist jedoch notwendig um

Zusammenhänge erkennen zu können. Semantikdefinitionen können im Allgemeinen alle Perspektiven umfassen weshalb es nicht möglich ist, Analysen und Miningschritte vorzuziehen oder auszulagern.

Aufbau von Instanzgraphen
Für das erstmalige Durchlaufen des Logs und die Generierung der Ereignislisten müssen sämtliche Anwendungsfälle komplett betrachtet werden, d.h. dieser Schritt hat eine Komplexität von $O(n \cdot m)$. Zur Generierung der Instanzgraphen wird die komplette Ereignisliste erneut durchlaufen und für jeden Instanzgraph der transitive Abschluss gebildet, wodurch die Prozesse einer Instanz im schlimmsten Fall noch einmal komplett durchlaufen werden müssen. Dies führt zu einem weiteren n und somit erfordert dieser Schritt $O(n^2 \cdot m)$.

Semantische Interpretation
Für jede vorhandene zweiwertige Semantikdefinition müssen sämtliche dualen Prozesskombinationen überprüft werden. Jede einwertige Regel muss für alle Prozesse überprüft werden. Bei der Überprüfung einer Regel werden alle Instanzgraphen durchlaufen (erfordert $O(n \cdot m)$). Insgesamt benötigt die semantische Interpretation somit:

Für zweiwertige Regeln: $O(n^2) \cdot O(n \cdot m) = O(n^3 \cdot m)$
Für einwertige Regeln: $O(n) \cdot O(n \cdot m) = O(n^2 \cdot m)$

Kapselung von Prozessen
Im Fall der funktionalen Perspektive werden Prozesshierarchien aufgebaut. Der erste Durchlauf der Kapselung von Prozessen, d. h. die Kapselung von rein elementaren Prozessen, erfordert wie die Überprüfung einer zweiwertigen Regel $O(n^3 \cdot m)$. Da jedoch Prozesshierarchien erstellt werden, muss die Kapselung von Prozessen im Allgemeinen häufiger durchlaufen werden. Im schlimmsten Fall wird pro Durchlauf ein elementarer Prozess gekapselt, weshalb ein weiteres n dazukäme und deshalb die Kapselung von Prozessen $O(n^4 \cdot m)$ benötigt.

Traversierung des Semantikgraphen und Überführung in ein Prozessmodell
Somit ist der Semantikgraph aufgebaut, welcher durch einfache Traversierung in ein Prozessmodell überführt werden könnte. Die Anzahl der Kanten im Graph entspricht der Anzahl der aufgefundenen zweiwertigen Regeln, was im Allgemeinen nicht vorhergesagt werden kann. Eine Worst-Case Abschätzung geht davon aus, dass nach der semantischen Interpretation ein *vollständiger* Graph vorliegt, d.h. jeder Knoten besitzt eine Beziehung zu jedem anderen Knoten. Der Graph enthält somit $n \cdot (n-1)$ Kanten. Das Traversieren des Graphen mittels Tiefensuche benötigt daher $O(n + n \cdot (n-1)) = O(n + n^2)$ im Worst-Case [35].

Es zeigt sich, dass die größte Komplexität dieses Verfahrens in der Kapselung und somit der Ausbildung der Prozesshierarchien besteht.

5.6 Überführung des Semantikgraphen in ein Prozessmodell

In den vorherigen Abschnitten wurde die Implementierung der semantischen Interpretation erläutert, welche als Ergebnis den Semantikgraphen liefert. Dieser Graph enthält alle Informationen die notwendig sind um ein Prozessmodell zu generieren. Voraussetzung ist jedoch das Vorhandensein eines entsprechenden Meta-Modells, welches die möglichen Elemente des Prozessmodells definiert. Anhand eines Meta-Modells werden den semantischen Regeln entsprechende Modellierungskonstrukte zugeordnet und deren Visualisierung definiert. Ein mögliches Framework zur Definition eines Meta-

Modells ist *oMME*. Das Open Meta Modelling Environment [36] ist eine Plattform zur Entwicklung, Bearbeitung und Visualisierung von Meta-Modellen. Sie basiert auf der Eclipse IDE und verwendet das Eclipse Modeling Framework (EMF).

Exemplarisch wird ein Meta-Modell definiert, welches elementare Prozesse als Kreise mit Prozessbezeichnern darstellt. Kapseln werden als Vierecke modelliert, welche die enthaltenen Prozesse bzw. Kapseln umschließen. Der *StrictControlFlowConnector* wird als gerichteter durchgezogener Pfeil modelliert. Der *BossConnector* wird als roter gerichteter Pfeil dargestellt. Empfehlungen werden als gestrichelter gerichteter Pfeil visualisiert. Werden Prozesse zusätzlich durch Attribute beschrieben, kann auch hierfür durch das Meta-Modell eine Visualisierung definiert werden. Ein exemplarisches Prozessmodell zeigt Abbildung 24. Im zu Grunde liegenden Meta-Modell wurde folglich definiert, dass das Attribut (d.h. die Semantik) „*Chefsache*“ durch einen grünen Kreis dargestellt wird. Anhand des Modells wird nun sofort klar, dass an Prozess D ein Chef beteiligt sein muss. Durch den *Dashed-Arrow* zwischen den beiden Kapseln (A, B) und (C, D) ist nun ersichtlich, dass die Reihenfolge der Ausführung der beiden Kapseln eigentlich irrelevant ist, jedoch auf Basis einer Empfehlung die visualisierte Reihenfolge beispielsweise vorteilhaft sein könnte. Der *BossConnector* zwischen C und D weist daraufhin, dass für „normale“ Mitarbeiter die angegebene Reihenfolge unumkehrbar ist, für einen Chef jedoch schon.

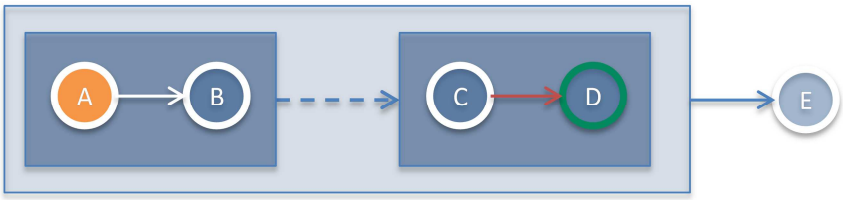

Abbildung 24: Resultierendes Prozessmodell auf Basis eines exemplarischen Meta-Modells

Die Vorgehensweise zur Überführung des Semantikgraphen in ein Prozessmodell, in Anbetracht eines vorhandenen Meta-Modells, ist wie folgt:

- Der Semantikgraph wird vollständig traversiert.
- Jeder besuchte Knoten wird als Prozess modelliert. Dabei wird die Attributes-Liste eines jeden Knotens durchlaufen und bei entsprechendem Vorhandensein eines Attributes das im Meta-Modell für das betrachtete Attribut definierte Modellierungskonstrukt an den Prozess angehaftet bzw. dessen Eigenschaften wie definiert verändert (Änderung der Farbe, Form etc.).
- Nun wird die funktionale Perspektive betrachtet. Für jede Kapsel wird ein entsprechendes Modellierungskonstrukt konstruiert (wie im obigen Fall Vierecke) und die enthaltenen Prozesse zugeordnet.
- Zuletzt wird für jede vorhandene Kante (dieser ist im Feld *EdgeType* die entsprechende Semantik hinterlegt) ein Konnektor im Prozessmodell erzeugt. Im Fall eines Kontrollfluss-Konnektors wird die Prozesshierarchie der beiden aktuell betrachteten Prozesse solange durchlaufen, bis sich die IDs unterscheiden. Zwischen diesen beiden Prozessen wird dann im Modell das Modellierungskonstrukt erzeugt.

  Beispiel in Anlehnung an Abbildung 24:
  Die generierte Prozesshierarchie von Prozess A aus dem Semantikgraphen lautet: 4, 3, 1, A
  Prozesshierarchie von Prozess E: 4, E

  Bei der Traversierung der Prozesse im Semantikgraphen werden aktuell die Prozesse A und E betrachtet. Zwischen ihnen wurde eine *StrictControlFlowConnector*-Semantik festgestellt. Dies ist eine Kontrollfluss-Semantik, deshalb werden deren Hierarchien nun durchlaufen bis sich die IDs

unterscheiden. Dies ist im Fall von HierarchyElement(A) = 3 und HierarchyElement(E) = E der Fall. Der *StrictControlFlowConnector* wird folglich zwischen Kapsel 3 und dem elementaren Prozess E im Modell konstruiert.

6 Zusammenfassung und Ausblick

Die vorliegende Arbeit befasst sich mit der Entwicklung eines neuartigen Process Mining Verfahrens, genannt *SemanticPM*, welches vorwiegend die Generierung einer allgemeingültigen Basis-Datenstruktur anstrebt, die ein flexibles Mining nach beliebig definierten Semantiken ermöglicht. Während bisher existierende Verfahren stets nur eine Perspektive der perspektivenorientierten Prozessmodellierung (POPM) separat betrachten und analysieren, strebt *SemanticPM* eine kombinierte Betrachtung und Einbeziehung sämtlicher Perspektiven von POPM an und ermöglicht damit das Auffinden von Zusammenhängen zwischen den einzelnen Perspektiven.

Der erste Schritt des Process Mining ist die Generierung von Instanzgraphen, ein Verfahren welches dem Ansatz von Agrawal et al. ähnelt. Diese beinhalten alle vorkommenden Prozesse, deren ausführende Agenten, verwendete Daten, verwendete Tools sowie deren Beziehungen zueinander innerhalb einer Instanz. Eine weitergehende Analyse wird durch die semantische Interpretation der Instanzgraphen durch *SemanticPM* erreicht. Anschließend liegt mit dem Semantikgraph eine Datenstruktur vor, welche sämtliche im Log auftretenden Prozesse als Knoten enthält. Anhand von vordefinierten Semantiken, welche *SemanticPM* als Input dienen, werden den Prozessen Eigenschaften (Attributes) zugewiesen und Relationen eines bestimmten Typs (Edges) zwischen den Prozessen dargestellt. Die aufzufindenden Semantiken können beliebig definiert und flexibel hinzugefügt oder entfernt werden. Die Relationen und Eigenschaften umfassen und kombinieren sämtliche Perspektiven des POPM. Hierdurch ermöglicht *SemanticPM* die Erkennung von Zusammenhängen zwischen verschiedenen Perspektiven. Die funktionale Perspektive (Erkennung von atomaren bzw. kompositen Prozessen) wird durch die den Knoten hinzugefügten Prozesshierarchien abgedeckt. Wurde zwischen zwei (elementaren oder kompositen) Prozessen keine Kontrollfluss-Semantik entdeckt, konnte eventuell auf Basis einer vordefinierten aber ebenso flexibel auswählbaren Bedingung eine Kontrollfluss-Empfehlung hinzugefügt werden. Durch Traversierung des Semantikgraphen kann bei Vorhandensein eines Meta-Modells ein Prozessmodell mit entsprechenden Modellierungskonstrukten generiert werden.

Das Gebiet des Process Mining auf Basis von Semantikdefinitionen lässt großen Spielraum für zukünftige Forschungs- und Entwicklungsaktivitäten. Als nächster Schritt sollte eine vollständige Integration in ein Meta-Modellierungsframework wie oMME [36] angestrebt werden. Denkbar wäre eine Implementierung, welche es ermöglicht, flexibel und unkompliziert Semantikdefinitionen in das System einzugeben und dieser Definition anschließend ein beliebiges, noch nicht belegtes, Modellierungselement zuzuweisen. Anschließend wird ein gegebenes Log nach den eingegebenen Semantiken durchsucht. Entsprechend der Zuweisung im Meta-Modell erfolgt schließlich automatisch eine Generierung von Prozessmodellen. Diese können anschließend in Diskussionsrunden analysiert und bei Bedarf remodelliert bzw. überarbeitet werden. Systeme wie oMME ermöglichen nach beendeter Modellierungsphase die direkte Implementierung des modellierten Prozesses in ein Workflow Management System (WfMS) und damit die systemgestützte Ausführung des Prozesses. Abbildung 25 zeigt die Einbindung in das Meta-Modellierungsframework oMME. Das zu analysierende Log wird dabei einerseits aus den aufgezeichneten Prozessen des WfMS ProcessNavigator generiert und andererseits aus den Logging Informationen der manuellen Prozessausführung mit Hilfe des Process Observer Tools [37].

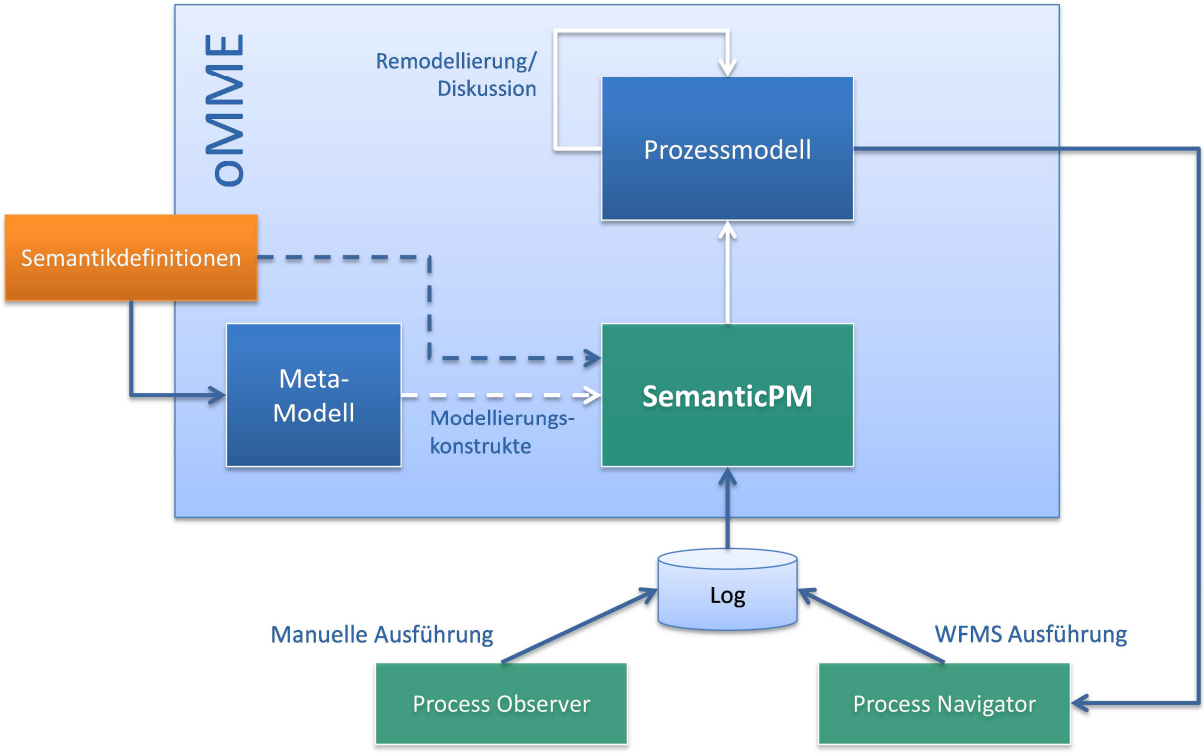

Abbildung 25: Vollständige Integration in ein Meta-Modellierungsframework

Process Mining auf Basis von expliziten Semantikdefinitionen trägt so einen großen Schritt dazu bei, den Business Process Lifecycle weiter zu automatisieren und so zeit- und kostenintensive Phasen zu verkürzen.

Literaturverzeichnis

[1] Zur Muehlen, M.; Ho, D.: *Risk Management in the BPM lifecycle*. Springer, New York City, USA, 2006.

[2] van der Aalst, W.; Weijters, T.; Maruster, L.: *Workflow mining: Discovering process models from event logs.* Knowledge and Data Engineering, IEEE Transactions, 1128-1142, 2004.

[3] vom Brocke, J.; Rosemann, M.: *Handbook on Business Process Management: Strategic Alignment, Governance, People and Culture*. International Handbooks on Information Systems, Springer, 2010

[4] Jablonski, S., Bussler, C.: *Worfklow Management – Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press, London, 1996.

[5] van der Aalst, W.; Weijters, A.: *Process-Aware Information Systems*. Kapitel 10, Process Mining, John Wiley & Sons Ltd, Seiten 235-255, 2005.

[6] Stock, W. G.; Stock, M.: *Wissensrepräsentation: Informationen auswerten und bereitstellen*. Oldenbourg Wissenschaftsverlag, 2008.

[7] van der Aalst, W.; Reijers, H.; Weijters, A.; van Dongen, B.; de Medeiros, A.; Song, M.; Verbeek, H.: *Business Process Mining: An Industrial Application.* In Information Systems, Bd. 32, Nr. 5, Seiten 713-732, 2007.

[8] Hitzler, P.; Krötzsch, M.; Rudolph, S.; Sure, Y.: *Semantic Web - Grundlagen*, Erste Auflage, Springer Verlag, 2008.

[9] W3C: Resource Description Framework (RDF): http://www.w3.org/RDF. Version 2004. 2004.

[10] W3C: Web Ontology Language (OWL): http://www.w3.org/OWL. Version 2004. 2004.

[11] Horrocks, J.; Patel-Schneider, P. F.; Boley, H.; Tabet, S.; Grosof, B.; Dean, M.: *SWRL: A Semantic Web Rule Language Combining OWL and RuleML.* W3C Member Submission, 2004.

[12] Reiter, R.: *On Closed World Data Bases.* University of British Columbia Vancouver, BC, Canada, 1977.

[13] Cook, J.E.; Wolf, A. L.: *Automating process discovery through event-data analysis*, 1995.

[14] van der Aalst, W.; Weijters, A.: *Process Mining: A research Agenda.* In Computers in Industry, Ausgabe 53(3), Seiten 231-244, 2004.

[15] van Dongen, B.; de Medeiros, A.; Verbeek, H.; Weijters, A; Reijers, H.; van der Aalst, W.: *The ProM Framework: A new era in process mining tool support.* In G. Ciardo und P.Darondeau, editors, 26th International Conference on Applications and Theory of Petri Nets, Seiten 444-454, LNCS 3536, 2005.

[16] Agrawal, R.; Gunopulos, D.; Leymann, F.: *Mining Process Models from Workflow Logs (erweiterte Fassung),* 1998.

[17] Hwang, S.-Y.; Yang, W.-S.: *On the discovery of process models from their instances*. Decision Support Systems, 2002.

[18] Golani, M.; Pinter, S.: *Discovering workflow models from activities' lifespans*. Computers in Industry, (53), 2004.

[19] Golani, M.; Pinter, S.: *Generating a Process Model from a Process Audit Log,* 2004.

[20] Lang, M.: *Prozess-Mining und Prozessbewertung zur Verbesserung klinischer Workflows im Umfeld bilderzeugender Fächer.* Dissertation, Universität Erlangen, 2008.

[21] van der Aalst, W.: *The Application of Petri Nets to Workflow Management.* In Journal of Circuits Systems and Computers, Citeseer, 1998.

[22] de Medeiros, A.; van Dongen, B.; van der Aalst, W.; Weijters, A.: *Process Mining: Extending the α-algorithm to Mine Short Loops,* BETA Working Paper Series WP 113, 2004.

[23] Wen, L.; Wang, J.; Sun, J.: *Detecting Implicit Dependencies between Tasks from Event Logs.* In Xiaofang Zhou et al. (Eds.), editor, Proceedings of the 8th Asia Pacific Web Conference (APWeb 06), Seiten 591–603. Springer, 2006.

[24] Weijters, A.; van der Aalst, W.; de Medeiros, A.: *Process Mining with the HeuristicsMiner Algorithm.* Beta Working Paper 166, Beta Research school for Operations Management and Logistics, 2006.

[25] de Medeiros, A.; Weijters, A.; van der Aalst, W.: *Genetic Process Mining: An Experimental Evaluation.* Journal of Data Mining and Knowledge Discovery, Bd. 14, Nr. 2, Seiten 245–304, 2006.

[26] van der Aalst, W.; Song, M.: *Mining Social Networks: Uncovering interaction patterns in business processes.* Business Process Management, Springer, 2004.

[27] Rozinat, A.; van der Aalst, W.: *Decision Mining in Business Processes.* BPM Center Report BPM-06-10, 2006.

[28] Günther, C.: *Dissertation über Semantikdefinitionen in der Prozessausführung*. Universität Bayreuth, Bayreuth, laufende Arbeit, voraussichtliche Publikation in 2012.

[29] van der Aalst, W. (editor): *Process Mining and Monitoring Processes and Services: Workshop Report.* In Leymann, F.; Reisig, W.; Thatte, S.; van der Aalst, W. (Hrsg.): The Role of Business Processes in Service Oriented Architectures, Nr. 6291 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), 2006.

[30] Günther, C.; Schönig, S.; Jablonski, S.: *Guidance Enhancement in Workflow Management Systems,* 27th Symposium on Applied Computing, 2012.

[31] Tarjan, R.: *Depth-first search and linear graph algorithms*. In: SIAM Journal on Computing. Bd. 1, Nr. 2, S. 146-160, 1972.

[32] Igler, M.; Moura, P.; Zeising, M.; Jablonski, S.: ESProNa: *Constraint-Based Declarative Business Process Modeling*. Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2010 14th IEEE International, 2010.

[33] Google: *google-gson - A Java library to convert JSON to Java objects and vice-versa*, http://code.google.com/p/google-gson, 2011.

[34] Naveh, B.: *JGraphT - a free Java graph library*, http://www.jgrapht.org, 2011.

[35] Heun, V.: *Grundlegende Algorithmen – Einführung in den Entwurf und die Analyse effizienter Algorithmen*, Zweite Auflage, Vieweg Verlag, 2003.

[36] Volz, B.: *Dissertation über neuartige Modellierungsparadigmen und das LMM*. Universität Bayreuth, Bayreuth, 2010.

[37] Schönig, S.; Günther, C.; Jablonski, S.: *Process Discovery through Process Mining Based on Manually Generated Logs*, voraussichtliche Publikation in 2012.