FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Process Mining: Design und Implementierung eines Ansatzes zur zeitnahen Integration von Ereignisprotokollen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Thomas Neuböck
Matrikelnummer 0525610

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Dr. Marco Zapletal

Wien, 31.08.2011

(Unterschrift Verfasser) (Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Thomas Neuböck
Brauhausstraße 1/28, 2351 Wiener Neudorf

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Abstract

Enterprise information systems (e.g.: ERP, CRM, WFM and B2B systems) store information about executed processes in log structures. These event logs, containing a massive amount of events, can be used for discovering business processes. Process Mining techniques extract process information out of the existing event logs, analyze and interpret the data and use it in corresponding algorithms for deriving process models. The resulting process models are visualized and presented to the decision makers, which are able to make further decisions based on this gained knowledge. The process mining algorithms deliver the process model in three different perspectives: (1) process perspective, (2) organizational perspective and (3) case-perspective. Additional application classes of Process Mining are the comparison of the gained process model to a given reference model or the extension of a given model with additive information.

Existing Process Mining techniques perform the extraction based on historical process information. These traditional Process mining application scenarios typically require manual extraction and transferring of the process data, which is usually complex and expensive. Such a proceeding is inapplicable, if the take-over of data shall be done frequently. Moreover, a real time monitoring of running processes - as operational support - isn't realizable, because the data actualization is too time expensive. A real time or near-real time integration of event-logs in existing Process Mining techniques would require to solve supplementary challenges, for example to find an efficient architecture to achieve a real time integration.

This master thesis discusses two approaches for using Process Mining in real time: (1) Real time Process Mining as operational support and (2) Using traditional process mining algorithms in a near-real time manner (the work is focused on this aspect). Furthermore, the work introduces an implementation for using traditional scenarios in a near-real time manner. The implemented architecture includes a Plugin for the ProM Framework, which converts and imports event-logs generated from the Apache Camel EAI Framework into ProM. The second implementation artifact is a mechanism for recording occurring events in the Apache Camel EAI Framework. The basic ideas as well as the used existing techniques are explained in the course of this master thesis. The resulting implementation of the designed architecture shows, whether a near-real time integration of event-logs for traditional process mining could be achieved and which further challenges have to be solved.

Kurzfassung

In Ereignisprotokollen betrieblicher Informationssysteme (z.B.: ERP, CRM, WFM und B2B Systeme) sind für Unternehmen bzw. Organisationen interessante und oft auch unbekannte Daten über die in den Systemen ablaufenden Prozesse enthalten, welche, wenn entsprechend interpretiert, genutzt werden können um die Prozesse und den notwendigen IT Support zu optimieren oder unbekannte Abläufe zu erkennen und diese zu verstehen. Process Mining Techniken extrahieren aus den vorhandenen Ereignisprotokollen der Informationssysteme Prozessinformationen, werten die gewonnen Daten mittels entsprechenden Algorithmen aus und visualisieren die Ergebnisse für die Entscheidungsträger in ansprechender und verständlicher Form. Die Analyse der gewonnenen Informationen kann in drei unterschiedlichen Perspektiven erfolgen: (1) Prozess-Perspektive, (2) Organisatorische-Perspektive sowie (3) Fall-Perspektive. Weiterführende Anwendungsklassen des Process Minings beschäftigen sich mit der Gegenüberstellung der ableitbaren Informationen mit einem Soll-Prozessmodell oder der Erweiterung eines solchen um zusätzliche Daten.

In den existierenden Process Mining Techniken werden aus den vorhandenen Ereignisprotokollen die Prozessinformationen aufgrund eines historischen Standes extrahiert, analysiert und für darauf basierenden Entscheidungen verwendet. Traditionelle Anwendungsszenarien erfordern meist einen aufwändigen manuellen Übernahmeprozess der extrahierten Prozessdaten zur Verwendung dieser in den entsprechenden Process Mining Techniken, was eine ineffiziente Anbindung der Datengrundlage an die verwendeten Techniken darstellt. Auch eine Überwachung sowie Beurteilung der Prozesse in Echtzeit – als operative Unterstützung - ist mit den bestehenden Techniken nicht möglich. Die daraus resultierenden Erkennungszeiten von prozessrelevanten Information folglich lang. Eine zeitnahe Integration der Ereignisprotokolldaten in bestehende Process Mining Techniken würde es erfordern zusätzliche Herausforderungen, beispielsweise eine effiziente Gestaltung der Gesamtlösungsarchitektur, zu meistern.

Im Zuge dieser Diplomarbeit werden die Ansätze einer zeitnahen Nutzung von traditionellen Anwendungsszenarien sowie von Process Mining in Echtzeit diskutiert, wobei die zeitnahe Nutzung im weiteren Verlauf dieser Arbeit vertiefend betrachtet wird. Unterstützend wird eine Implementierung eines ProM Framework PlugIn‘s zur Verarbeitung von Ereignisprotokollen des Apache Camel EAI Framework sowie die hierfür gewählte Architektur vorgestellt. Die dieser Implementierung zu Grunde liegenden Ideen und die dabei verwendeten Techniken, werden an den relevanten Stellen im Detail erklärt. Das Ergebnis der Implementierung soll zeigen, ob und wie die zeitliche Lücke zwischen dem traditionellen Ansatz des Process Minings und einem Ansatz einer zeitnahen Integration von Ereignisprotokollen in Process Mining Techniken geschlossen werden kann und welche Herausforderungen hierfür gemeistert werden müssen.

Inhaltsverzeichnis

KAPITEL 1

Einführung

Heutzutage werden in vielen Unternehmen bzw. größeren Organisationen elektronische Informationssysteme genutzt, um Geschäftsprozesse innerhalb der Organisation zu unterstützen und in Folge zu vereinfachen beziehungsweise zu optimieren. Dies kann einerseits die Unterstützung von einfacheren Back-office Prozessen als auch die Umsetzung komplexer zwischenbetrieblicher Prozesse betreffen. Bei den am verbreitetsten Informationssystemen handelt es sich beispielweise um Technologien wie Enterprise Resource Planning (ERP) zur Unterstützung der Ressourcenplanung, Customer Relationship Management (CRM) zur systematische Gestaltung der Kundenbeziehungs-Prozesse, Workflow Management Systeme (WfM) zur Steuerung von automatisierten Prozessen, Enterprise Application Integration (EAI) zur unternehmensweiten Integration der Geschäftsfunktionen verteilt über unterschiedliche Applikationen, Plattformen und Services, in welchen unterschiedliche Prozesse automatisiert abgebildet werden können. Diese Technologien werden eingesetzt um innerhalb des Unternehmens die notwendigen Prozesse zu unterstützen. Ebenfalls ermöglicht diese Automatisierung die Optimierung des Prozessablaufs, welcher über unzählige Akteure verteilt erfolgen kann. Üblicherweise läuft solch ein einzelner, zusammenhängender Prozess, innerhalb eines Informationssystems (IS), über einen längeren Zeitraum. Die Durchführung der einzelnen Aktivitäten kann durch unterschiedliche Personen erfolgen oder automatisch abgebildet sein und folglich durch das System selbst abgewickelt werden. Diese IS-unterstützten Geschäftsprozesse werden naturgemäß, aufgrund des zum Zeitpunkt der Umsetzung vorhandenen Prozesswissens, implementiert. Basis für diese Umsetzung ist üblicherweise ein Soll-Modell des entsprechenden Geschäftsprozesses. Im Laufe der Zeit muss aufgrund der Dynamik der Umwelt oder Veränderungen innerhalb des Unternehmens die Prozessabwicklung bei Bedarf angepasst werden. Diese Adaptionen der Prozesse können zu Abweichungen vom ursprünglichem Soll-Modell sowie zu folglich nicht optimalen Prozessbläufen führen. Der Fokus einer Prozessautomatisierung wird üblicherweise auf die Abbildung der notwendigen Prozesse innerhalb der Organisation gelegt. Die Berücksichtigung der Dynamik und Flexibilität der implementierten Prozesse sowie der Bereitstellung einer Möglichkeit zur Prozess- oder Systemdiagnose ablaufender Prozesse, wird meist nur wenig Bedeutung zu

gemessen.

Typischerweise bieten Informationssysteme Funktionalität zur Protokollierung der einzelnen durchgeführten Anwendungsfälle und folglich der zugehörigen Aktivitäten. Die Granularität sowie der enthaltene Detailgehalt dieser Protokollierungen unterliegt keinen einheitlichen Richtlinien, somit können diese Charakteristiken zwischen unterschiedlichen Informationssystemen große Unterschiede aufweisen. Die protokollierten Informationen der Prozessausführung, d.h. einzelne Prozessinstanzen (Case), können entsprechend für weitere Analysen und Logiken verwendet werden. Eine Prozessinstanz besteht selbst aus einzelnen Teilaktivitäten (Tasks). Abhängig vom jeweiligen Informationssystem können nur gewisse Ereignistypen im Ereignisprotokoll berücksichtigt werden. Viele Informationssysteme enthalten in ihren Ereignisprotokollen auch Informationen zu den beteiligten Akteuren der einzelnen Aktivitäten. Üblicherweise sind in komplexen Geschäftsprozessen unterschiedliche Personen und/oder das System selbst (automatisierte Tätigkeiten) an der Ausführung der einzelnen Teilaktivitäten beteiligt. Dies stellt im Zuge der Analyse der bestehenden Prozesse einen interessanten Aspekt für weitere Entscheidungen dar.

Process Mining nutzt diese Protokolle der ausgeführten Aktivitäten sowie die darin enthaltenen Informationen als Basis für die Durchführung unterschiedlicher Anwendungsszenarien. Mit geeigneten Techniken und Algorithmen werden aufgrund dieser Basis relevante Informationen und Wissen extrahiert, Analysen durchgeführt, Modelle verschiedener Perspektiven generiert, die resultierenden Ergebnisse visualisiert oder aufgrund des gewonnenen Modells ein Vergleich, eine Erweiterung oder eine Verfeinerung eines Soll-Modells durchgeführt. In den existierenden Process Mining Lösungen werden diese Ereignisprotokollen auf Basis eines historischen Standes verarbeitet und für darauf basierende Entscheidungen und Aktionen verwendet. In bestehenden Umsetzungen werden den entsprechenden Process Mining Techniken diese historischen Prozessdaten vorwiegend in einer nicht automatisierten Art und Weise zur Verfügung gestellt. Sollte ein anderer Datenstand analysiert werden ist dieser Schritt folglich erneut manuell durchzuführen (was natürlich auch Zeit- und Kosten-intensiv ist). Dieser Datenübernahmeprozess ist folglich meist ineffizient und hat eine zeitliche Abweichung zwischen der Protokollierung von Ereignissen und der Nutzung in den Process Mining Techniken zur Folge. Eine Verbesserung dieses Übernahmesystematik der Prozessdaten kann zu einer effizienteren Nutzung (z.B.: in dynamischen Umgebungen) von traditionellen Anwendungsszenarien führen, indem dieser Übernahmeprozess automatisiert und folglich beschleunigt wird. Weitere Anwendungsfälle, wie die Überwachung der Prozesse in Echtzeit, Vorhersage der zu erwartenden Restzeit des Prozesses oder der Vorschlag nächster Aktionen zur Ermöglichung eines optimalen Prozessablaufs, sind mittels des Ansatzes auf Basis von historischen Daten nicht möglich. Solche Anwendungsszenarien wären durch eine Echtzeitanbindung und Nutzung der Ereignisprotokolldaten durch Unterstützung geeigneter Techniken theoretisch umsetzbar.

Im weiteren Verlauf dieser Diplomarbeit werden sowohl der Ansatz einer operativen Unterstützung durch Process Mining als auch die Möglichkeit einer zeitnahen Nutzung von traditionellen Anwendungsszenarien diskutiert. Weiterführend werden zusätzliche Thematiken und

Herausforderungen, welche sich für diese Einsatzzwecke ergeben, erklärt und entsprechende Lösungsansätze präsentiert. Im Speziellen werden die Herausforderungen sowie die notwendigen Prämissen hinsichtlich einer Nahezu-Echtzeitanbindung von Ereignisprotokollen an Process Mining Techniken vertiefend behandelt. Ergänzend zu diesen theoretischen Überlegungen wird eine Implementierung einer zeitnahen Anbindung eines Ereignisprotokolls an traditionelle Process Mining Techniken, welche im Zuge der Diplomarbeit entwickelt wurde, vorgestellt und die daraus gewonnen Schlüsse diskutiert.

Methodisches Vorgehen

Als Methodisches Vorgehen im Zuge dieser Diplomarbeit wurde der Ansatz *Design Science* [1] gewählt. *Design Science* ist ein problemlösungsorientierter Ansatz, wobei hierbei keine natürlich vorkommende Phänomene, sondern durch Menschen zur Zielereichung geschaffene untersucht werden. Als Design wird eine bewusste Organisation von Ressourcen zur Erreichung der deklarierten Ziele verstanden. Mittels *Design Science* werden organisatorische Probleme identifiziert, wobei das Ziel entweder ungelöste Probleme oder eine effizientere Lösung für bereits gelöste Probleme adressieren kann. Im Zuge des Methodischen Vorgehens *Design Science* werden IT Artefakte (z.B.: diese können sowohl Software oder mathematische Modelle darstellen) geschaffen und diese in weiterer Folge evaluiert. Dies bedeutet, dass die entstandenen Artefakte in einen funktionierenden System eingesetzt werden können und illustrieren damit die Machbarkeit. Erbringt die Evaluierung der Artefakte ein negatives Resultat wird der gewählte Lösungsansatz verworfen ansonsten kann dieser als Lösung des adressierten Problems angenommen werden.

In [1] werden sieben Richtlinien für die Durchführung von *Design Science* beschrieben:

- **Design als zielgerichtetes Artefakt:** Die Lösung ist ein innovatives und zielgerichtetes Artefakt.
- **Problemrelevanz:** Ziel ist die Entwicklung von technischen Lösungen für aktuelle und zukünftige Probleme. Die Relevanz wird über den Nutzen definiert, welche eine Lösung erbringt.
- **Evaluierung:** Nutzen, Qualität und Effizienz müssen evaluiert werden.
- **Beitrag der Forschung:** *Design Science* muss einen klar definierbaren Beitrag zur Problemlösung und/oder den Forschungsmethoden liefern.
- **Methodische Strenge in den Forschungsmethoden:** Im Zuge der Schaffung und Evaluierung von Artefakten ist eine methodische Strenge einzuhalten.
- **Design als Suchprozess:** Design stellt einen Suchprozess dar, in welchem Lösungen vorgeschlagen, verfeinert und evaluiert werden.
- **Weitergabe von Forschungsergebnissen:** Die Ergebnisse aus der Durchführung müssen verständlich und effektiv vermittelt werden.

Struktur der Diplomarbeit

Die Diplomarbeit ist wie folgt aufgebaut:

- In Kapitel 2 wird Process Mining inklusive dessen Basiskonzepte sowie die unterschiedlichen Perspektiven und Anwendungsszenarien vorgestellt. Des Weiteren wird ein Vorgehensmodell für die Umsetzung von Process Mining Implementierungen, angelehnt an ein Data Mining Prozessmodell, vorgeschlagen und beschrieben. Abschließend werden Herausforderungen, welche im Besonderen für diese Diplomarbeit relevant sind, beleuchtet und das Process Mining Framework ProM vorgestellt.

- Kapitel 3 erklärt die Thematik einer zeitnahen Nutzung von traditionellen Process Mining Anwendungsszenarien sowie des Echtzeit Process Minings und dessen Anwendungsszenarien. Es werden im Weiteren die in der Literatur bekannten sowie weitere mögliche Herausforderungen beleuchtet.

- In Kapitel 4 wird eine im Zuge der Diplomarbeit umgesetzte Lösung zur zeitnahen Anbindung von Ereignisprotokollen an Process Mining Techniken vorgestellt. Die entwickelte Lösung setzt sich aus zwei eigenständigen Komponenten zusammen: a) Protokollierungsmechanismus und b) Import- und Konvertierungsmechanismus. Beide Komponenten sowie alle relevanten Technologien, auf welche diese Implementierungen aufsetzen, werden im Zuge dieses Kapitels erläutert.

- In Kapitel 5 werden ausgewählte Process Mining Algorithmen innerhalb eines bestehenden Process Mining Frameworks ProM demonstriert. Als Datengrundlage werden Prozessdaten einer eigens umgesetzten Beispielanwendung verwendet. Diese Anwendung nutzt die im Zuge dieser Diplomarbeit umgesetzte Lösung zur Erzeugung dieser Daten.

- Abschließend werden in Kapitel 6 die Ergebnisse und Erkenntnisse dieser Diplomarbeit sowie der praktischen Umsetzung einer Lösung zur zeitnahen Anbindung von Prozessdaten an Process Mining Techniken zusammengefasst.

KAPITEL 2

Process Mining

2.1 Allgemein

Der Begriff Process Mining umfasst Techniken und Methoden um aus den Ereignisprotokolldaten ausgeführter Geschäftsprozesse Prozesswissen, über die im Zeitverlauf getätigten Ausführungen, zu extrahieren und aus diesen Informationen Prozessmodelle zu generieren. Process Mining verfolgt einen ähnlichen Grundgedanken wie Data Mining: die Extraktion von Wissen aus bestehenden Daten. Data Mining verfolgt das Ziel aus vorhandenen Geschäftsdaten, welche in verschiedenen Online-Transaction-Processing Systemen (OLTP) erzeugt und gespeichert wurden, neuartige und gültige Muster zu extrahieren. Diese Muster können in Folge genutzt werden um, unter Zuhilfenahme von speziellen Techniken, unbekanntes Wissen über die Geschäftstätigkeiten des Unternehmens abzuleiten und diese Erkenntnisse als Basis für weitere Entscheidungen zu nutzen. Data Mining kann wie folgt definiert werden [14]:

> *"Data Mining is a collection of techniques for efficient automated discovery of previously unkown, valid, novel, useful and understandable patterns in large databases. The patterns must be actionable so that they may be used in an enterprise's decision making process."* [14]

Beispiel für Data Mining Anwendungen sind Association Rule-Mining, zur Ermittlung von kausalen und relevanten Zusammenhängen zwischen Objekten (z.B.: Warenkorb-Analyse), die Klassifizierung von Daten in vorhandene Gruppierungen (Supervised Learning) sowie Cluster-Analyse (Unsupervised Learning - z.B.: Segmentierung und Gruppierung von Daten in sich logisch ergebende Verbunde). Konkrete Ziele des Data Minings sind es aus vorhandenen Geschäftsdaten die Voraussage und Planung der Geschäftstätigkeiten zu erleichtern, Kundenbeziehungen und die -kommunikation zu optimieren oder die Erkenntnisse zur Analyse von Ausreißern, beispielsweise zur Betrugserkennung, zu verwenden. Aufgrund der ähnlichen Intention der beiden Techniken, werden im Zuge dieser Arbeit Analogien zwischen Data Mining und Process Mining gezielt aufgezeigt und besprochen.

Process Mining versucht, analog wie Data Mining, durch den Einsatz von Methoden und Techniken eine Extraktion von Wissen durchführen. Process Mining fokusiert sich hierbei jedoch auf die Prozessperspektive, d.h. es werden die einzelnen Aktionen der ablaufenden Prozesse sowie deren Attribute und die beteiligten Akteure in den Mittelpunkt des Interesses gestellt. Es wird versucht, aufgrund des vorhandenen Prozesswissens, Analysen durchzuführen sowie gültige Prozessmodelle abzuleiten. Ziel ist es mittels solch einer Prozesswissenextraktion für die Organisation neuartige, gültige, unbekannte sowie nützliche Modelle zu generieren und aus diesen Wissen und Erkenntnisse zu gewinnen, um dieses Wissen für weitere Entscheidungen als Grundlage bzw. Ausgangsbasis zu verwenden.

Fragestellungen, welche durch den Einsatz von Process Mining Techniken beantwortet werden können, sind beispielsweise:

- Wie werden die einzelnen Prozesse aktuell ausgeführt?
- Entspricht die aktuelle Ausführung dem definierten Modell?
- Was sind die meist frequentierten Pfade des Prozessmodells?
- Wie viele Personen sind an der Ausführung einzelner Prozesse beteiligt?
- Wie sehen die Kommunikationsstruktur und Abhängigkeiten zwischen den einzelnen Akteuren aus?
- Sind Muster hinsichtlich der Delegation von Arbeit zu erkennen?
- Gibt es Ressourcen-Engpässe innerhalb des Prozessmodells?

Angelehnt an die Definition von Data Mining [14] lässt sich Process Mining wie folgt definieren:

> *Process Mining ist eine Sammlung von Techniken für die automatische und effiziente Extraktion und Ableitung von bisher unbekannten, gültigen, nützlichen und verständlichen Prozessmodellen auf Basis von Prozessdaten aus Ereignisprotokollen. Diese Prozessmodelle müssen ausreichend relevant sein, um sie als Grundlage für weitere Entscheidungen verwenden zu können.*

Grundlegende Aspekte des Process Minings sind daher die Ereignisprotokolle als Ausgangsbasis, welche durch Informationssysteme erzeugt werden, sowie Modelle die aus den Ereignisprotokollen abgeleitet werden (Abbildung 2.1). Diese Aspekte sowie Themen und Erkenntnisse aus der vorhandenen Process Mining Literatur [33] [35] [32] [22] [20] [31] [3] [5] [17], werden in den nächsten Absätzen nach folgender Gliederung besprochen:

- Grundbegriffe: Ereignisprotokoll, Prozessinstanz und Ereignis
- Perspektiven: Prozess-, Organisatorische- und Case-Perspektive
- Klassen: Discovery, Conformance, Refinement und Extension

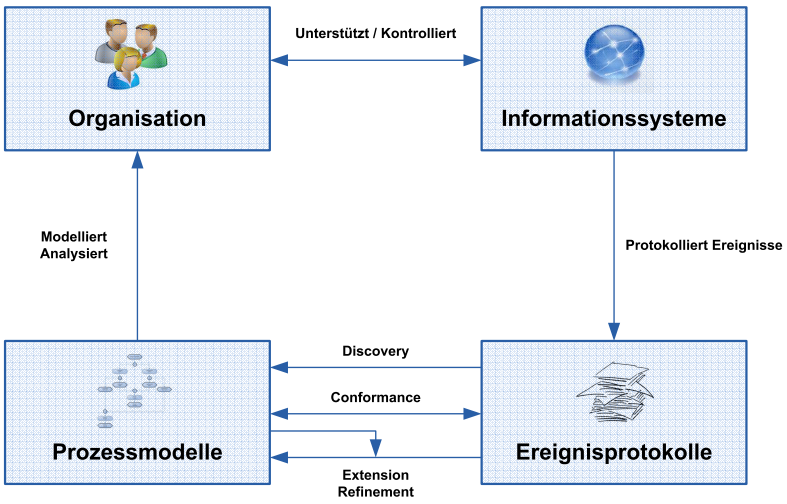

Abbildung 2.1: Process Mining: Konzeptionelle Übersicht [50]

- Process Mining Vorgehensmodell
- Herausforderungen im Zuge des Process Minings

2.2 Grundbegriffe

Zentrale Grundbegriffe des Process Minings sind Ereignisprotokolle, Prozessinstanzen sowie Ereignisse. Ereignisprotokolle stellen das gesamte Prozesswissen eines Informationssystems dar, d.h. die Menge aller ausgeführten Prozessinstanzen innerhalb dessen. Prozessinstanzen stellen eine konkrete Ausführung eines Prozesses dar. Zwischen den einzelnen Aktivitäten innerhalb solch einer Prozessinstanz besteht ein zeitlicher Zusammenhang, aufgrund dessen die einzelnen Aktivitäten in eine eindeutige Ordnung gebracht werden können. Diese eindeutige Ordnung ergibt eine Sequenz der Ereignisse der jeweiligen Prozessinstanz. Diese Sequenz muss jedoch nicht für alle Prozessinstanzen eines implementierten Prozesses ident sein. Um den zeitlichen Zusammenhang zwischen den Ereignissen einer Prozessinstanz zu ermitteln, wird in den meisten Fällen das Konzept der Verwendung von Zeitstempeln angewendet. Steht kein Zeitstempel der einzelnen Ereignissen zur Verfügung, müssen andere Indikatoren (z.B.: Nummerierung) zur Sotierung der Ereignisse verwendet werden. Diese Informationen stellen jedoch oftmals den zeitli-

chen Zusammenhang zwischen den Ereignissen nur teilweise korrekt dar. Ein Ereignis innerhalb des Protokolls bildet die Aktivität innerhalb einer Prozessinstanz ab. Solch ein Ereignis kann von einer natürlichen Person oder automatisiert, durch das System selbst, durchgeführt werden und einen spezifischen Ereignisstatus, abhängig vom jeweiligen System, annehmen (z.B.: Gestartet, Beendet, Abgebrochen). Üblicherweise werden in einem Protokoll folgende Informationen, welche für die Nutzung von Process Mining Techniken notwendig sind, für ein Ereignis abgelegt:

- Case ID: Eindeutige Prozessinstanzbezeichnung
- Ereignis ID: Eindeutige Ereignisbezeichnung
- Ereignistyp: Zustand einer spezifischen Aktivität
- Ressource: Für die Ausführung verantwortlicher Akteur
- Zeitstempel: Zeitliche Information zur Aktivitätsausführung
- Sonstige Informationen: Weitere Detaildaten zur Aktivität

In 2.1 ist der Aufbau solch eines Ereignisprotokoll illustrativ dargestellt. In den folgenden Ausführungen werden die abgebildeten Informationen als Ausgangsbasis für vertiefende Beispiele genutzt.

Prozessinstanz	Ereignis ID	Ereignistyp	Ressource	Zeitstempel	Sonstiges
Case 1	Task A	Abgeschlossen	Thomas	12-04-2011 17:59:05	...
Case 2	Task A	Abgeschlossen	Thomas	12-04-2011 19:12:05	...
Case 1	Task B	Abgeschlossen	System	13-04-2011 05:00:01	...
Case 1	Task C	Abgeschlossen	Michaela	13-04-2011 10:30:44	...
Case 1	Task D	Abgeschlossen	Thomas	13-04-2011 12:12:05	...
Case 1	Task E	Abgeschlossen	Michael	13-04-2011 10:30:44	...
Case 2	Task B	Abgeschlossen	System	14-04-2011 05:01:01	...
Case 2	Task D	Abgebrochen	Thomas	15-04-2011 15:11:48	...
Case 2	Task C	Abgeschlossen	System	19-04-2011 05:00:01	...
Case 2	Task E	Abgeschlossen	Michael	19-04-2011 10:20:33	...
Case 2	Task E	Abgebrochen	Michael	20-04-2011 13:02:09	...
Case 3	Task A	Abgeschlossen	Thomas	20-04-2011 15:30:44	...
Case 4	Task A	Abgeschlossen	Thomas	20-04-2011 15:45:17	...
Case 3	Task B	Abgeschlossen	System	21-04-2011 05:00:01	...

Tabelle 2.1: Beispiel: Ereignisprotokoll

Ereignisprotokolle stellen wie bereits erwähnt die Ausgangsbasis für den Einsatz der Process Mining Techniken dar. Es werden, für den betroffenen Zeitraum (abhängig vom Umfang des Ereignisprotokolls oder der konkreten Aufgabenstellung kann das Protokoll zeitlich abgegrenzt und somit nur ein Teilausschnitt des Prozesswissens betrachtet werden), alle Prozessinstanzen

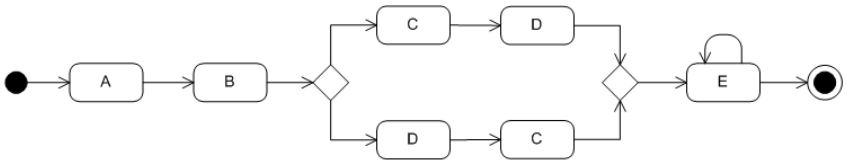

Abbildung 2.2: Process Mining: Aktivitätsdiagramm zu den Beispieldaten aus Tabelle 2.1

und folglich alle darin enthaltenen Ereignisse betrachtet, um aufgrund dieser Informationen entsprechende Prozessmodelle zu generieren. Ein Prozessmodell ist eine abstrakte Repräsentation der real ablaufenden Prozesse aufgrund Basis des bestehenden Wissens. Diese Prozessmodelle ergeben sich aufgrund der Menge der Prozessinstanzen des Ereignisprotokolls, d.h. der eindeutigen Sequenzen von Ereignissen. Für das Prozessmodell können aufgrund unterschiedlichen Ereignisreihenfolgen innerhalb der Prozessinstanzen auch parallele Aktivitätsabläufe oder Schleifen der Ereignisse abgeleitet werden. Betrachtet man die Prozessinformationen aus dem Ereignisprotokoll 2.1 ist der Schluss auf folgende Aussagen möglich und das abgebildete Prozessmodell 2.2 ableitbar.

- Task A ist immer das erste Ereignis des Prozesses
- Task A wird immer vom Akteur Thomas ausgeführt
- Task B wird immer vom Akteur System ausgeführt
- Nach Task B kann entweder Task C oder Task D ausgeführt werden (Parallelität)
- Der Prozess kann im Task D manuell abgebrochen werden (Prozessinstanz *Case 2*)
- Task E kann beliebig oft, am Ende einer Prozessinstanz, ausgeführt werden

2.3 Perspektiven

Das Ergebnis der Process Mining Techniken ist ein Prozessmodell basierend auf den vorhandenen Ereignisprotokollen. Abhängig von den verwendeten Analyse- und Mining-Algorithmen kann das Prozessmodell unterschiedliche Perspektiven und Aspekte, des im Informationssystem umgesetzten Prozesses, beleuchten bzw. aufzeigen. Die Prozessmodelle, welche man im Zuge des Process Minings aus den Ereignisprotokollen gewinnen kann, lassen sich nach [35]

[22] in drei unterschiedliche Perspektiven gliedern: a) Process-Perspektive, b) Organisatorische-Perspektive und c) Case-Perspektive.

2.3.1 Prozess Perspektive

Die dominanteste Perspektive ist die Prozess-Perspektive, welche die Betrachtung des Kontrollflusses des jeweiligen Prozesses ermöglicht. Hierzu muss der Kontrollfluss zwischen den Aktivitäten innerhalb des jeweiligen Prozesses abgeleitet werden, indem die Ereignisse der einzelnen Prozessinstanzen aufgrund einer logischen und zeitlichen Abfolge geordnet dargestellt werden. Eine zeitliche Abfolge kann zum Beispiel durch die Verwendung eines Zeitstempels der Ereignisse ermittelt werden. Die Betrachtung des Prozessmodells in der Prozess Perspektive ergibt sich aus der Menge an vorhandenen Prozessinstanzen und kann, wie bereits erwähnt, aufgrund unterschiedlicher Abfolgen der Aktivitäten in Prozessinstanzen, auch Parallelitäten oder Schleifen enthalten. Prozessmodelle können beispielweise als Petri Netz [24] oder als ereignisgesteuerte Prozesskette (EPK) [18] dargestellt werden. In der Abbildung 2.3 sind die Prozessdaten aus dem gegebenen Beispiel (Tabelle 2.1) in Form eines Petri Netz dargestellt. Ein klassischer Process Mining Algorithmus, zur Extraktion eines Modells in der Prozess Perspektive, ist der α - Algorithmus [31]. Weitere Beispiele solcher Algorithmen sind der Genetic Algorithmus [21], der Heuristic Algorithmus [53] oder der ILPMiner [37], welche verschiedene Ansätze zur Extraktion von Modellen verfolgen.

2.3.2 Organisatorische Perspektive

Die organisatorische Perspektive stellt die beteiligten Akteure innerhalb der Prozesse und deren Verbindung zueinander dar. Zur Vereinfachung der Darstellung wird meist versucht die konkreten Akteure der Prozesse in verschiedene Rollen zu klassifizieren. Die Beteiligten können im Fall der organisatorischen Perspektive einerseits natürliche Personen, andererseits auch Sub-Systeme – d.h. technische Akteure – sein. Diese Akteure können für die Ausführung der Aktivität verantwortlich gewesen sein, das Ergebnis der Ausführung erhalten haben oder in einer anderen Form am entsprechendem Ereignis beteiligt gewesen sein. Diese sozialen Zusammenhänge können beispielsweise in einem sozialen Netzwerk dargestellt werden, wobei hier unterschiedliche Blickwinkel (z.B.: Kommunikations- oder Delegationsstruktur) im Mittelpunkt des Interesses stehen können. Üblicherweise liegt der Ausführung von Prozessen implizit das Prozesswissen zugrunde, d.h. die Akteure wissen was, wann, wie und wo erledigt werden muss. Ziel einer Organisation ist es, die implementierten Informationssysteme so personenunabhängig wie möglich zu gestalten, um einen höheren Flexibilitätsgrad der Organisation zu erreichen und allgemeine organisatorische Probleme zu vermindern. Das abgeleitete Prozessmodell kann Rückschlüsse auf das vorhandene Prozesswissen erlauben. Dies ermöglicht es in vielen Fällen, aufgrund des Process Mining Ergebnisses, zentrale Schlüsselpositionen in den umgesetzten Prozessen zu erkennen und diese Erkenntnisse für strategische Entscheidungen zu berücksichtigen. Beispiele für Algorithmen zur Extraktion von Modellen der organisatorischen Perspektive sind der Social Network Miner [30], der Organizational Miner [28] oder der Role Hierarchy Miner [13]. In der Abbildung 2.3 sind die Prozessdaten aus dem gegebenen Beispiel (Tabelle 2.1) in einer organisatorischen Struktur sowie in einem sozialen Netz dargestellt.

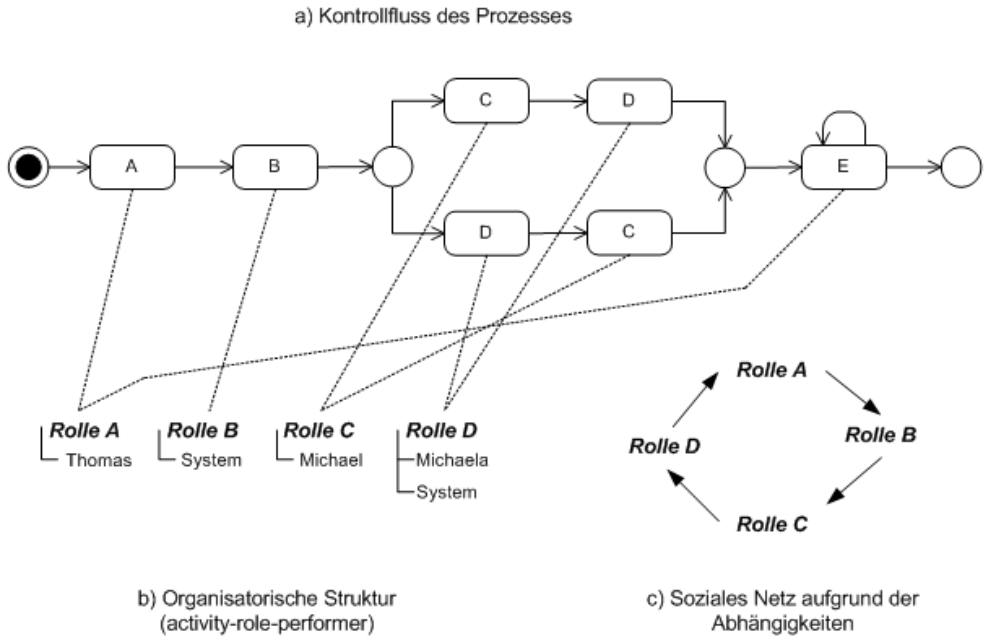

Abbildung 2.3: Unterschiedliche perspektivische Betrachtung der Beispielprozessdaten aus Tabelle 2.1. a) Kontrollfluss innerhalb der Prozesse (Prozessmodell) sowie b) und c) Organisatorische Perspektive der Beispielprozessdaten. [35]

2.3.3 Case Perspektive

In manchen Fällen sind Aussagen bzw. Informationen aus den Ereignisprotokolldaten ableitbar, welche weder in der Prozess noch in der organisatorischen Perspektive entsprechend berücksichtigt werden oder darstellbar sind. Solche spezifischen Aspekte der Prozessdaten können in einer eigenen Perspektive - der Case Perspektive - entsprechend aufbereitet werden. Es gibt eine große Zahl an Eigenschaften eines Prozesses, welche für eine Analyse interessant sind. Beispielsweise Informationen zur Ausführungszeit der einzelnen Aktivitäten, zur Ermittlung des Minimums, Maximums oder Durchschnitts, oder zur Erkennung potentieller Engpässe im aktuellen Prozess. Die meisten Algorithmen bedienen sich hierfür zusätzlicher Prozessinformationen von einzelnen protokollierten Aktivitäten, um entsprechende Analysen durchzuführen. Ein möglicher Anwendungsfall ist die Ermittlung der Relation zwischen Entscheidungen in der Prozessdurchführung und spezifischen Ereignisattributen, was beispielsweise mit dem Decision Miner [25] umgesetzt wurde. Mittels des Decision Miners wird untersucht, wie entsprechende Attribute der vorliegenden Prozessdaten einen Einfluss auf die Entscheidung der Akteure haben. Es bestehen eine große Anzahl von unterschiedlichen untersuchbaren Eigenschaften, wie beispielsweise die maximale, minimale oder durchschnittliche Ausführungszeit oder die Erkennung von Engpässen

innerhalb der Prozessdurchführung. Ein entsprechendes Ergebnis kann wichtige Erkenntnisse zeigen (Warum wurde in manchen Fällen entsprechend gehandelt?), die für strategische Entscheidungen oder für die Verbesserung der Prozesse eingesetzt werden können.

2.4 Klassen

Wie in der Einführung bereits erwähnt, ermöglichen es die Methoden des Process Minings unterschiedliche Anwendungsszenarien abzudecken. Alle Process Mining Techniken versuchen Daten über die ablaufenden Prozesse aus einem Ereignisprotokoll zu extrahieren, um daraus Informationen zu gewinnen bzw. Modelle zu generieren. Die einzelnen Anwendungsszenarien lassen sich aufgrund der grundlegenden Motivation und dem angestrebten Ziel in vier Klassen gliedern: Discovery, Conformance, Refinement und Extensions [22] [34]. Folgend wird ein kurzer Überblick über die Eigenschaften und Ziele dieser einzelnen Process Mining Klassen gegeben.

2.4.1 Discovery

Ziel ist es aus den vorhandenen Ereignisprotokolldaten ein Ist-Prozessmodell der implementierten Prozesse abzuleiten, ohne hierfür ein Soll-Prozessmodell (a-priori Modell) zu berücksichtigen. Solch ein Anwendungsszenario macht meist dann Sinn, wenn die umgesetzten Prozesse nicht explizit geplant wurden (z.B.: Top-Down Implementierung), sondern im Laufe der Zeit entstanden und gewachsen sind. In diesem Fall unterstützt Discovery von Prozessmodellen die Organisation bzw. deren Entscheidungsträger dabei, Prozesse und etwaige kritische Pfade, Engpässe oder Gründe für Ausnahmen im Zuge der Durchführung interner Prozesse besser zu verstehen bzw. zu erkennen. Dies ist die traditionelle Anwendungsklasse des Process Minings, welche mit einer großen Anzahl an Forschungsarbeiten belegt ist. Für das Anwendungszenario Discovery wurden bereits viele Methoden und Algorithmen umgesetzt und in einem praktischen Umfeld evaluiert. Beispiele für solche Algorithmen sind der α-Algorithmus [31], Genetic Algorithmus [21], der Heuristic Algorithmus [53] oder der ILPMiner [37]. Auch die in den organisatorischen Perspektiven beschriebenen Algorithmen sind für solch ein Anwendungsszenario anwendbar.

2.4.2 Conformance

Anders als beim Anwendungsszenario Discovery besteht bereits ein vorhandenes Soll- Prozessmodell (a-priori Modell), welches im Zuge des Conformance Anwendungsszenarios eine zentrale Rolle spielt. Ziel ist es das bestehende a-priori Prozessmodell mit dem aus den Ereignisprotokolldaten gewonnenen Ist-Prozessmodell auf Übereinstimmung zu überprüfen. Beispielweise kann dieses beschreibende Prozessmodell (Soll-Modell) als Diagramm aus einem einfachen Prozessmodellierungstool oder im Falle eines ERP Systems als Referenzmodell vorliegen. Dieser Soll-Ist Vergleich wird in der Literatur als Delta Analyse bezeichnet. Mittels der Delta Analyse können etwaige Unterschiede sowie Unstimmigkeiten zwischen den zwei Prozessmodellen entdeckt und beschrieben werden, sowie die zu erwartenden Auswirkungen aufgrund dieser Abweichungen bemessen werden. Conformance ist üblicherweise dann interessant, wenn die umgesetzten Prozesse im Zuge einer Top-Down Planung durch das Management festgelegt wurden

und die ausführende Ebene diese Definitionen als entsprechende Prozesse umgesetzt hat. Conformance kann durch die Gegenüberstellung des Soll- und des Ist-Modells Fehler im Zuge der Beschreibung, Kommunikation oder Umsetzung aufzeigen und nimmt folglich einen Kontrollcharakter an.

Beispielsweise können in einem Soll-Prozessmodell die Veränderungenen innerhalb des Unternehmens oder auch der Umwelt und folglich in den jeweiligen Geschäftsprozessen nicht berücksichtigt werden und das Ist-Modell somit ein aktuelleres Bild der Prozessausführung zeigen. Andererseits kann der Grund solcher Abweichungen auch in einem falschen Verständnis der jeweiligen Geschäftsprozesse (d.h. ein fehlerhaftes Soll-Modell) oder einer fehlerhaften Implementiertung der durch das Soll-Prozessmodell definierten Geschäftsprozesse liegen. Aktuell bestehen nur eine geringe Anzahl von technischen Lösungsansätzen zur effizienten Analyse von Prozessmodellabweichungen. Eine der bedeutendsten Arbeiten [25] zu diesem Anwendungsszenario beschreibt Methoden und Metriken zur vollständigen Überprüfung der Übereinstimmung zwischen bestehenden und abgeleiteten Modellen. In [22] wird ein weiterer Ansatz zur Prüfung der Übereinstimmung von Modellen diskutiert.

2.4.3 Refinement

Das Anwendungsszenario Refinement geht ebenfalls von einem bestehenden Soll-Prozessmodell (a-priori Modell) aus. Anders als bei der Conformance erwartet man zwischen den zwei Modellen keine Übereinstimmung, sondern man rechnet mit bestehenden Abweichungen oder Inkonsistenzen. Refinement soll diese Inkonsistenzen ermitteln und das bestehende Soll-Modell aufgrund der gewonnen Erkenntnisse ergänzen bzw. aktualisieren. Das Anwendungsszenario Refinement nimmt somit einen korrigierenden Charakter an. Solche eine Anwendung macht Sinn, wenn sich die ursprüngliche Idee der Umsetzung nicht mit dem wirklich ablaufenden Prozessen deckt oder wenn sich seit der geplanten Implementierung dieser Prozesse, sonstige Abläufe innerhalb der Organisation oder die Umwelt entsprechend stark verändert haben und somit die Übereinstimmung mit dem ursprünglichen Modell nicht mehr gegeben ist. In [22] werden Methoden beschrieben, welche aufgrund von Conformance Analyse Techniken, Inkonsistenzen zwischen Soll- und Ist-Modellen bereinigen, indem das Soll-Modell entsprechend aktualisiert bzw. angepasst wird.

2.4.4 Extension

Wie bei den Anwendungsszenarien Conformance und Refinement, geht man auch bei der Klasse Extension von einem bestehenden Soll-Prozessmodell (a-priori Modell) aus. Anders als bei den vorhergehenden Anwendungsklassen, besteht das Ziel nicht darin, die Übereinstimmung der Modelle zu überprüfen, sondern ein bestehendes Soll-Modell mit zusätzlichen Informationen zu ergänzen und anzureichern. Solche Informationen können zusätzliche Daten oder Perspektiven der ablaufenden Prozesse darstellen. Somit nimmt das Anwendungsszenario Extension einen ergänzenden Charakter an. Ein Beispiel für eine Technik dieser Anwendungsklasse ist der Decision Miner [25], mit welchem analysiert wird ob und welche Attribute Auswirkungen auf gewisse Entscheidungen im Zuge der Prozessausführung haben.

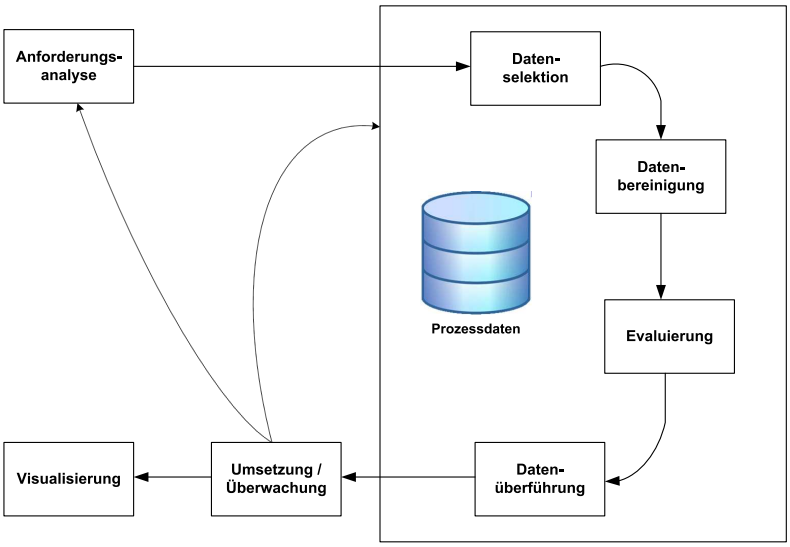

Abbildung 2.4: Process Mining: Vorgeschlagenes Vorgehensmodell

2.5 Vorgeschlagenes Vorgehensmodell

Um aufgrund der Daten eines Ereignisprotokolls Prozessmodelle ableiten zu können, ist es notwendig sich einige grundsätzliche Fragen zu stellen: Welche Aussagen sollen aus den Prozessdaten abgeleitet werden können? Welche Informationssysteme liefern entsprechende Protokolldaten? Enthalten die Einträge in diesen Protokollen alle Informationen die ich für das Process Mining brauche? Wie müssen die Daten aufbereitet werden, um mit vorhandenen Algorithmen die Daten analysieren zu können?

Folgend wird versucht, angelehnt an den Data Mining Prozess [14], ein grundlegendes Vorgehen für die Umsetzung von Process Mining zu skizzieren (Abbildung 2.4). Dieser Prozess mag in manchen Situationen nicht ausreichend sein, es soll aber gezeigt werden, wie die Grundidee hinter einer Umsetzung aussehen sollte. Das Vorgehensmodell umfasst mehrere Teilphasen, wobei zwischen diesen in manchen Anwendungsfällen zusätzliche Iterationen notwendig sein können. In Folge werden die Teilphasen des vorgeschlagenen Modells im Detail beschrieben.

2.5.1 Anforderungsanalyse

Die Entscheidungsträger müssen Ziele formulieren, welche mittels der Durchführung von Process Mining erreicht werden sollen. Naturgemäß ist es zu erwarten, dass mit einem klar definierten Ziel, die Qualität der Ergebnisse deutlich höher ist, als ohne grundlegenden Plan. Es ist folglich auch deutlich einfacher und transparenter, die Ergebnisse, welche man im Prozess erzielt hat, mit den geplanten Erwartungen zu vergleichen. Für manche spezifischen Anforderungen werden sich gewisse Techniken besser eignen und durch die gezielte Auswahl der notwendigen Datenquellen können Probleme in der weiteren Umsetzung vermieden werden. Im Zuge einer Anforderungsanalyse sollten natürlich die Auftraggeber, d.h. die Hauptinteressenten hinsichtlich der Ergebnisse, sowie die zur Umsetzung Beauftragten beteiligt sein, um ein gemeinsames Verständnis der Anforderung und der gewünschten Ziele zu schaffen.

2.5.2 Datenselektion

Es muss entschieden werden, welche Datenquelle, d.h. im Fall von Process Mining welches Ereignisprotokoll, die notwendigen Prozessdaten enthält. Das entsprechende Ereignisprotokoll bzw. das betroffene Quellsystem sollte sich aufgrund der Ergebnisse der Anforderungsanalyse automatisch ergeben oder relativ einfach ableiten lassen. Im einfachsten Fall stehen die Ereignisprotokolle in einer atomaren Form, d.h. ein Quellsystem mit einem einzigen Ereignisprotokoll, zur Verfügung. Die Datenselektion erschwert sich jedoch, wenn das Ereignisprotokoll in mehreren unterschiedlichen Teilprotokollen vorliegt oder wenn sich die aus der Anforderung ergebenden Prozesse sich über mehrere Informationssysteme erstrecken. Im zweiten Fall spricht man üblicherweise von heterogenen Quellsystemen. Die daraus resultierenden Konsequenzen werden in weiterer Folge noch im Detail diskutiert.

2.5.3 Aufbereitung der Ereignisprotokolldaten

Sollte das Ereignisprotokoll in einer geeigneten Form, z.B.: in einer Protokolldatei vorliegen, stellt sich die Prozessdatenaufbereitung als tendenziell einfach dar. Besteht jedoch das Ereignisprotokoll aus mehreren Tabellen, macht es meist Sinn diese Protokolle in ein einzelnes aggregiertes Protokoll zusammenzufassen und für die weitere Analyse diese Aggregation als Ausgangsbasis zu verwenden. Gleiches gilt, wenn man einen Prozess über mehrere Informationssysteme betrachtet. Dieser Vorgang kann sehr ressourcen-intensiv sein, beginnend bei der Analyse, Planung und der konkreten Aufbereitung der Prozessdaten. Eine weitere Thematik im Zuge der Aufbereitung ist die Erweiterung des Ereignisprotokolls, um für die geplante Analyse notwendige Attribute. Mögliche Probleme im Zuge dessen, sind beispielsweise das Fehlen eines Indizes zur eindeutigen Sortierung der Ereignisse innerhalb von Prozessinstanzen oder das Fehlen eines Indikators zur Zuordnung von Ereignissen zu konkreten Prozessinstanzen. Oft können diese Problematiken zu notwendigen Anpassungen der Protokollierung bestehender Informationssysteme führen, falls diese Informationen nicht über andere Wege aus den Daten automatisch abgeleitet und ergänzt werden können. Eine automatische Ergänzung und Erweiterung der Prozessdaten im Nachhinein, würde eine naturgemäß hohe Fehlerwahrscheinlichkeit mit sich bringen, was, wenn möglich, zu vermeiden ist. Die erwähnten Herausforderungen an die Datenaufbereitung

und folglich an die Durchführung des Process Minings werden im folgenden Kapitel noch im Detail behandelt.

2.5.4 Evaluierung der Process Mining Technik

Sobald die Daten in geeigneter Form vorliegen, müssen passende Process Mining Techniken evaluiert werden, um mit den vorhandenen aufbereiteten Prozessdaten, das im Zuge der Anforderungsanalyse definierte Ziel zu erreichen. Die Ergebnisse der Evaluierung pro vorhandener Technik können gegenübergestellt werden, wobei hier natürlich die Prüfung der Signifikanz der Ergebnisse im Vordergrund steht. Zusätzlich sollte auch die Performance und Robustheit der einzelnen Techniken beachtet werden. Dieser Evaluierungsprozess kann in sich iterativ erfolgen. Sollte sich im Zuge der Evaluierung zeigen, dass die aufbereitenden Daten für eine Technik in einer ungeeigneten Form vorliegen, muss die Iteration nochmals zurück zur Aufbereitung der Daten führen. Ergebnis dieser Evaluierung sind die notwendigen Algorithmen und Techniken (falls passende vorhandene Tools und Frameworks existieren können diese weiterverwendet werden), welche man zur Erreichung der definierten Ziele einsetzen kann.

2.5.5 Konvertierung der Daten

Aufgrund des Ergebnisses der Evaluierung der notwendigen Techniken zur Zielerreichung, kann eine Überführung der Ereignisprotokolldaten in ein entsprechendes Format notwendig sein. Dies ist beispielsweise erforderlich, wenn man sich Techniken bedient, welche in existierenden Tools oder Frameworks umgesetzt sind. Solche Tools und Frameworks (z.B.: ProM Framework) erwarten die Ereignisprotokolldaten meist in einer spezifischen Form. Bedient man sich keines vorhandenen Process Minings Tools, sondern entscheidet sich für die Implementierung eigener Process Mining Techniken, kann der Schritt einer expliziten Konvertierung in den meisten Fällen entfallen. Eine Konvertierung könnte jedoch auch in diesem Fall für eine strukturierte Weiterverarbeitung Sinn machen. Sollte ein Konvertierungsschritt notwendig sein, ist es erforderlich diesen entsprechend in der Durchführung zu berücksichtigen. Dies kann, falls die Datenquelle bzw. das Quellformat unterstützt werden, bedeuten, dass diese Konvertierungen direkt durch das gewählte Process Mining Tool übernommen werden kann. Ansonsten müssen eigene Konvertierungsmethoden umgesetzt und in den Prozess integriert werden. Praktisch kann diese Konvertierung der Daten im Zuge der Datenaufbereitung erledigt werden, theoretisch sollte jedoch eine bewusste Trennung zwischen, Aufbereitung und Konvertierung in ein entsprechendes Format, gehalten werden.

2.5.6 Implementierung/Beobachtung

Aufgrund der getroffenen Entscheidungen ist die Implementierung mit den gewählten Techniken oder Tools sowie mit eventuell notwendigen Aufbereitungen und Konvertierungen durchzuführen, um die daraus resultierenden Ergebnisse für die Entscheidungsträger zu erzeugen. Die Ergebnisse sollten im weiteren Verlauf, d.h. im Zuge des laufenden Betriebes, regelmäßig auf Effizienz und Aussagekraft geprüft werden. Veränderungen im System, der Organisation oder der Umwelt, sowie im Umfang der Prozessdatenmenge (im Normalfall ist von einem Wachstum

auszugehen) können dazu führen, dass die gewählten Methoden nicht mehr die Ergebnisse in gewünschter Form oder Qualität erbringen oder die gewählten Techniken nicht mehr in benötigter Effizienz agieren. Sollten diese Veränderungen die bestehende Implementierung zu stark beeinträchten, können Anpassungen der bestehenden Lösung oder eine erneute Evaluierung geeigneter Methoden erforderlich machen. Massive Veränderungen führen, wenn notwendig, entsprechend zu einem radikalen Redesign der Vor- und Aufbereitungsschritte.

2.5.7 Visualisierung der Ergebnisse

Die Aufbereitung der Ergebnisse für die Entscheidungsträger ist, wie im Data Mining Prozess, ein äußerst wichtiger Schritt. Die vorhandenen Process Mining Tools bieten Funktionalitäten um die Ergebnisse (z.B.: Prozessmodelle) mit verschiedensten Techniken entsprechend zu visualisieren. Die Visualisierung komplexer, umfangreicher Modelle stellt jedoch eine Herausforderung dar, welche im weiteren Verlauf diskutiert wird.

Der laufende Process Mining Prozess wird vor allem eine automatisierte Aufbereitung und Konvertierung der Daten sowie die Durchführung der Mining Algorithmen und Visualisierung der Ergebnisse umfassen. Äußerst wichtig ist die laufende Beobachtung und Beurteilung der implementierten bzw. eingesetzten Techniken und der daraus resultierenden Ergebnisse. Gegebenenfalls ist aufgrund von dieser Beobachtung die Entscheidung zu treffen, den laufenden Prozess entsprechend zu verändern oder zu erweitern.

2.6 Herausforderungen im Zuge des Process Minings

Process Mining macht es erforderlich einige komplexe Herausforderungen im Zuge des Planungs- und Durchführungsprozesses zu meistern. Wie bereits im Laufe der Erläuterung eines möglichen Vorgehensmodells besprochen, können abhängig vom aktuellen Problemkontext unterschiedliche Thematiken berücksichtigt und behandelt werden. Es kann erforderlich sein das Prozesswissen aus verschiedenen heterogenen Quellen abzuleiten, die Prozessdaten entsprechend aufzubereiten, diese Prozessdaten in unterschiedliche Perspektiven entsprechender Modelle zu überführen sowie die Darstellung des aus dem Prozess resultierenden Modells verständlich und adäquat erfolgen zu lassen. In [33] werden bekannte Herausforderungen vorgestellt, welche im Zuge der Anwendung Process Minings zu erwarten sind. Die nachfolgend beschriebenen Herausforderungen sollen die Komplexität des Process Minings aufzeigen und einen Überblick der bekannten Herausforderungen und möglichen zugehörigen Lösungsansätzen vermitteln. Des Weiteren werden auf Basis von [33] Thematiken, welche im Zuge dieser Diplomarbeit relevant sind, in den folgenden Absätzen aufgearbeitet.

Die meisten Process Mining-Algorithmen gehen von vollständigen und korrekten Quellprotokolldaten aus, um daraus die resultierenden Prozessmodelle zu generieren. Diese theoretischen Annahmen sind jedoch in vielen Informationssystemen nicht sichergestellt, d.h. man kann in vielen Fällen nicht von einem vollständigen Ereignisprotokoll für die Durchführung der Algorithmen ausgehen. Grund hierfür können beispielsweise Anpassungen oder Erweiterungen im Informationssystem sein, welche in der Protokollierung nicht berücksichtigt wurden. Möglicherweise

wurde auch bei der Implementierung des Informationssystems der Ereignisprotokollierung nur wenig Wert beigemessen. Dies kann dazu führen, dass nur ein Bruchteil der zur Ausführung vorhandenen (und für die Durchführung von Process Mining wichtigen) Informationen in den Protokollierungsmechanismus aufgenommen wurden. Folglich muss man sich zur Durchführung von Process Mining Techniken besonders zu folgenden Fragestellungen Gedanken machen und Ansätze entwickeln um mit solchen Problemfällen umgehen zu können:

- Wie kann mit fehlerhaften Prozessdaten („noise“) umgegangen werden?
- Wie können Unvollständigkeiten in den Prozessdaten berücksichtigt werden?
- Wie können versteckte Ereignisse aus den Prozessdaten abgeleitet werden?
- Wie können Prozessdaten aus heterogenen Informationssystemen als Ausgangsbasis verwendet werden?
- Wie kann mit fehlenden oder unvollständigen Verbindungen zwischen Anwendungsfällen und einzelnen Ereignissen umgegangen werden?
- Wie können resultierende Modelle adäquat präsentiert werden?
- Gibt es in den Prozessen Schleifen und wie können diese im Prozessmodell berücksichtigt werden?
- Welche Techniken können zur Delta-Analyse eingesetzt werden und mit welchen Problemen ist zu rechnen?

2.6.1 Berücksichtigung von fehlerhaften Prozessdaten („noise“)

Die Annahme, dass die Daten in einem korrekten Zustand (valide, vollständig, etc.) vorliegen, ist wie bereits erwähnt, in vielen Anwendungsfällen nicht gegeben. Es ist jedoch aus theoretischer Sicht notwendig diese Prämisse ausreichend gut zu erfüllen, um eine geeignete Grundlage für die Verwendung der unterschiedlichen Mining Algorithmen zu bieten (siehe auch Ableiten von versteckten Ereignissen). Oftmals kann es jedoch passieren, dass einzelne Prozessschritte/Ereignisse nicht erfasst werden oder die Ereignisse im Protokoll nicht in der tatsächlichen Reihenfolge persistiert werden. Da diese zeitlichen Beziehungen zwischen den einzelnen Ereignissen entscheidend für das errechnete Modell und der Robustheit der verwendeten Techniken sind, müssen solche Abweichungen bzw. Ausreißer fehlerhafter Erfassungen entsprechend berücksichtigt werden. Um eine entsprechende Robustheit zu gewährleisten, muss der Mining Algorithmus solche fehlerhaften Protokollierungen für die Generierung des Prozessmodells ausschließen. Dies kann zum Beispiel durch Berücksichtigung von Grenzwerten erfolgen, welche inkorrektes Verhalten oder Ausnahmen innerhalb des Ereignisprotokolls neutralisieren. Die Schwierigkeit einer solchen Lösung besteht in der Ermittlung bzw. Festlegung solcher Grenzwerte. Die Möglichkeiten erstrecken sich über die Bereitstellung von Grenzwerten als Eingabeparameter der Algorithmen oder durch Berücksichtigung heuristischer Ansätze [54] zur automatischen Ermittlung. Die Bereitstellung von Eingangsparametern erfordert naturgemäß das Domänenwissen des Benutzers, was für komplexe Prozesse beispielsweise nicht oder nur selten zu bewerkstelligen ist.

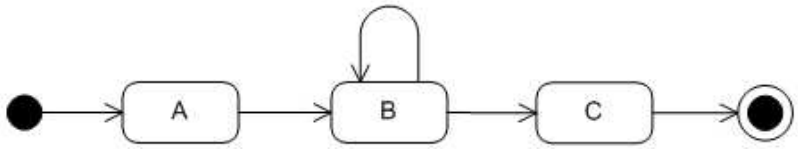

Abbildung 2.5: Process Mining: Schleifen

2.6.2 Berücksichtigung von unvollständigen Prozessdaten

Ähnlich dem Thema der inkorrekten Daten birgt die Verwendung von unvollständigen Prozessdaten die Gefahr ungültige Modelle zu generieren. Betrachtet man zum Beispiel das Ereignisprotokoll nur für einen bestimmten Zeitraum, d.h. nur einen Bruchteil des gesamten Ereignisprotokolls, kann es vorkommen, dass man Verhalten/Routen, welche mit einer geringen absoluten Häufigkeit auftreten, nicht im abgeleiteten Modell berücksichtigt. Dies wirkt sich bei Ereignisprotokollen mit vielen selten durchgeführten Prozesspfaden stärker aus, als bei Systemen mit einigen wenigen hochfrequentierten Prozesspfaden. Lösungsansätze verwenden Heuristiken um dieses Problem in den Griff zu bekommen. Diese Heuristiken basieren meist auf Occam's Ökonomieprinzip: „Bestehen zwei konkurrierende Theorien welche die gleichen Prognosen stellen, ist immer die einfachere Theorie die Bessere".

2.6.3 Ableiten von Schleifen

Eine Prozessinstanz kann die mehrfache Ausführung einer spezifischen Aktivität umfassen. Tritt ein entsprechendes Ereigniss innerhalb einer Prozessinstanz mehrfach auf (z.B.: unmittelbar hintereinander), führt dies typischerweise zu einer Schleife im abzuleitenden Modell (siehe 2.5). Eine mögliche Ereignissequenz für das in der Abbildung illustrierte Prozessmodell wäre ABC, ABBC, ABBBC und so weiter, d.h. das Ereignis B kann beliebig oft in einer Prozessinstanz vorkommen. Weiters können solche Schleifen innerhalb eines Prozesses auch eine Rücksprungmöglichkeit zu einer bereits durchgeführten Aktivität innerhalb des Prozesses darstellen. Die Thematik von Schleifen im Prozessmodell steht mit dem Ableiten von doppelten Ereignissen in enger Beziehung. Würden beispielsweise alle im Ereignisprotokoll vorhanden Prozessinstanzen mit einer zweimaligen Durchführung des Ereignisses A beginnen, könnte dies im Prozessmo-

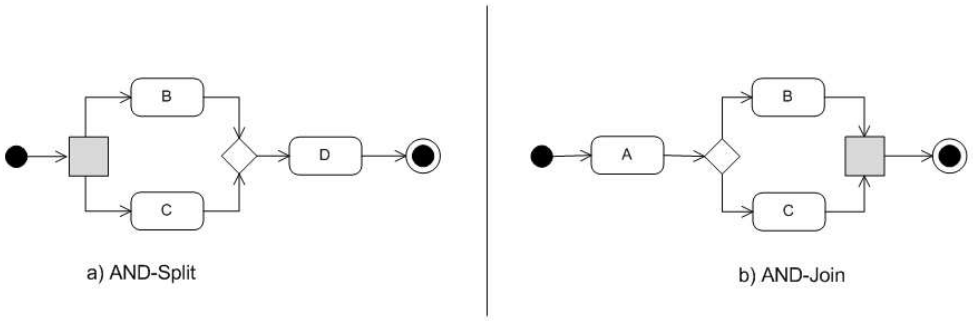

Abbildung 2.6: Process Mining: Versteckte Ereignisse

dell sowohl als Schleife dargestellt werden als auch als doppeltes Auftreten des Ereignisses A. Abhängig von der gewählten Process Mining Technik, können die resultierenden Prozessmodelle unterschiedlich dargestellt werden. Viele der Techniken treffen hierfür eigene Annahmen über Schleifen oder doppelte Ereignisse in Prozessmodellen, welche eine direkte Auswirkung auf die Korrektheit sowie die Vergleichbarkeit der abgeleiteten Prozessmodelle (im Besonderen bei komplexen Prozessmodellen) haben können.

2.6.4 Ableiten von versteckten Ereignissen

Die theoretische Annahme des Process Minings geht davon aus, dass jede aufgetretene Aktivität innerhalb eines Prozesses im jeweiligen Ereignisprotokoll mitgeschrieben wird. Sichergestellt ist diese Annahme jedoch in den meisten Informationssystemen nicht. Naturgemäß können nur Ereignisse die auch protokolliert wurden wieder aufgefunden werden. Alle nicht protokollierten Ereignisse folglich nicht. Theoretisch ist es jedoch in manchen Fällen möglich, solche versteckte Aktivitäten automatisch aus dem vorhandenen Protokoll abzuleiten und dem daraus resultierenden Prozessmodell hinzuzufügen. Werden beispielweise die Ereignisse B sowie C aufgrund des Ereignisprotokolls immer vor einem Ereignis D ausgeführt und ist die Reihenfolge von diesen zwei Ereignissen variabel, kann man von einer Parallelität zwischen diesen Ereignissen ausgehen. Folglich kann angenommen werden, dass vor diesen Ereignissen eine logische UND-Aufteilung (AND-Split) erfolgt (siehe Abbildung 2.6), was bedeutet, dass ein verstecktes Ereignis kann aus den Daten gefolgert werden. Andererseits kann diese Annahme auch genutzt werden, um eine logische UND-Vereinigung abzuleiten. D.h. wenn zum Beispiel B und C immer die letzten Ereignisse aller Prozessinstanzen sind, kann geschlossen werden, dass hier eine logische Zusammenführung in einem versteckten Ereignis besteht. Diese Erweiterung des abgeleiteten Prozessmodells um versteckte Ereignisse, kann beispielsweise nützlich sein, wenn

man das gefunde Modell zur Gegenüberstellung mit einem beschreibenden Soll-Prozessmodell verwenden will (Soll-Ist Vergleich).

2.6.5 Ereignisprotokolle aus heterogenen Informationssystemen

Wie bereits im oben beschriebenen Vorgehensmodell angesprochen, stellt die Aufbereitung der Prozessdaten im Generellen, sowie die Situationen in welchen die Ereignisprotokolle über mehrere heterogene Informationssystem verteilt vorliegen, im Speziellen, eine große Herausforderung in der Durchführung dar. Liegen die Prozessdaten in einem homogenen System vor, ist es im weiteren Sinne nicht relevant ob diese in einer oder in mehreren Strukturen abgelegt sind. Die Aufbereitung der vorliegenden Ausgangsprotokolle in eine zentrale Darstellung (z.B.: Verknüpfung mehrerer Tabellen in einer Sicht), als Grundlage für das Process Mining, birgt aufgrund einheitlicher Terminologien und Strukturen meist keine großen Probleme hinsichtlich der Komplexität. Sind jedoch systemübergreifende Prozesse von Interesse, sind einige Probleme im Zuge der Datenaufbereitung zu lösen.

Es muss sichergestellt werden, dass die vorhandenen Prozessinformationen von einer gemeinsamen Terminologie ausgehen, sprich es muss gewährleistet werden, dass die einzelnen Anwendungsfälle und Ereignisse den gleichen Informationsgehalt aufweisen. Dies würde beispielsweise bedeuten, dass wenn Informationssystem A in den Ereignissen Informationen zu der verantwortlichen Ressource protokolliert hat, dies in Informationssystem B ebenfalls berücksichtigt sein muss (falls die Prozessdaten zur Betrachtung aus der organisatorischen Perspektive herangezogen werden). Ebenfalls ist es als wichtig zu erachten, dass die Ereignisprotokolle eine zumindest ähnliche Informationsgranularität aufweisen. Dies ist besonders dann wichtig, wenn die Relevanz aller Systeme, welche im Zuge des Prozesses betroffen sind, ähnlich ist.

Ein weiteres schwieriges Thema stellt die Verknüpfung der einzelnen Ereignisse, aus den Ereignisprotokollen der einzelnen heterogenen Informationssysteme, zu einer Prozessinstanz dar. Dieses Problem kann umgangen werden indem sichergestellt wird, dass systemübergreifend eine eindeutige Super-Prozess-ID vergeben wird, welche den einzelnen Ereignissen in jedem System zugeordnet ist. Solche Annahmen werden in den meisten Fällen jedoch nicht durchführbar sein, da die Systeme nicht veränderbar sind oder von unterschiedlichen Herstellern entwickelt wurden und eine Anpassung zu teuer oder gar nicht möglich ist. Somit muss im Zuge der Aufbereitung versucht werden diese Verknüpfungen der Ereignisse aus den einzelnen Informationssystemen abzuleiten, was in sich selbst erneut ein komplexer eigenständiger Prozess ist. Solch ein Prozess würde kontext-abhängigen Input von Domänenexperten erfordern, um entsprechende Regeln und Überleitungen für eine automatische Durchführung zu definieren und umzusetzen.

Zusätzlich besteht das Thema der Sicherstellung der korrekten Reihenfolge der Ereignisse innerhalb der Prozessinstanzen, wenn sich diese über mehrere Informationssysteme erstrecken. Üblicherweise wird der Zeitstempel der aufgetretenen Ereignisse zur Ermittlung der Reihenfolge dieser innerhalb der Prozessinstanz verwendet. Unterschiedliche Informationssysteme können jedoch auch unterschiedliche Charakteristiken hinsichtlich der Protokollierung aufweisen oder verschiedene Strategien verfolgen. Beispielsweise könnte ein System die aufgetretenen Er-

eignisse sofort in das Ereignisprotokoll schreiben und ein anderes System die Protokollierung der Ereignisse verzögert durchführen (z.B.: mehrere Ereignisse werden auf einmal persistiert). Übernimmt in solch einem Fall nicht das System, beim Ereignisauftritt, die Aufgabe den aktuellen Zeitstempel festzuhalten, kann eine problematische Konstellation nicht kompatibler Zeitinformationen entstehen. Dies kann bei unterschiedlichen Verzögerungszeiten beim Schreiben der Ereignisprotokolldaten zu Verfälschungen des Gesamtprotokolls führen. Sicherzustellen ist, dass die Protokollierungsmechanismen immer mit einer annähernd ähnlichen Verzögerung in die jeweiligen Ereignisprotokolle schreiben, um eine Verfälschung der realen Reihenfolge der Aktivitäten zu vermeiden. Eine entsprechend verfälschte Reihenfolge, führt nach Anwendung von entsprechenden Algorithmen mit hoher Wahrscheinlichkeit zu einem fehlerhaften Prozessmodell. Übernehmen die Informationsysteme die Aufgabe den Zeitstempel direkt beim Ereignisauftritt festzuhalten, unabhängig von der jeweiligen Protokollierungsstrategie, können verfälschte Ereignisreihenfolgen verhindert werden.

Im Normalfall können die beschriebenen Problematiken mehrerer heterogener Informationsquellen durch die Umsetzung eines Data Warehouse [14] gelöst werden. Solch ein zusätzlicher Prozessschritt sollte in einem eigenen Teilprojekt umgesetzt werden. Im Besonderen sollte der Verknüpfung der einzelnen Prozesse große Bedeutung und folglich auch eine hohe Aufwendung von Ressourcen beigemessen werden. Oftmals wird es auch erforderlich sein die Protokollierung der einzelnen Informationssysteme im Vorhinein aufeinander abzustimmen. Dies bedeute zwar in den meisten Fällen einen einmalig erhöhten Aufwand in der Vorbereitung, kann jedoch die Kosten der Aufbereitung sowie die Fehlerwahrscheinlichkeit im Zuge des laufenden Betriebes deutlich senken.

2.6.6 Unvollständige Verbindungen zwischen Prozessinstanz und Ereignissen

Wie bereits im Zuge der Thematik der heterogenen Protokolldatenquellen beschrieben, besteht die Problematik, dass aufgrund der vorhandenen Informationen des Ereignisprotokolls, nicht in allen Fällen oder auch gar nicht, eine eindeutige Zuordnung eines Ereignisses zu einer konkreten Prozessinstanz möglich ist. Dieses Problem wird im Besonderen in Situationen mit heterogenen Quellsystemen häufig auftreten, ist aber auch in Fällen, bei welchen nur ein homogenes System als Quelle der Protokolle verwendet wird, möglich. Dieses Szenario ist mit einer nicht vernachlässigbaren Wahrscheinlichkeit denkbar und sollte bei der Planung entsprechend berücksichtigt werden. Beispielweise hat man im Ereignisprotokoll vier Ereigniseinträge mit einen jeweiligen Zeitstempel: 1 - Task A, 2 - Task B, 3 - Task A und 4 - Task B. Ohne der Information zur Prozessinstanz-ID bzw. den Zusammenhang zwischen den Verbindungen, kann man nun aus den vorhandenen Prozessdaten keine eindeutige Zuordnung durchführen. Man müsste diese Abhängigkeiten nun zum Beispiel manuell durchführen lassen oder die Daten im Detail analysieren, um herauszufinden ob aufgrund der Daten eine gültige Verbindung zwischen den Ereignissen ableitbar ist. In den meisten Fällen wird es sinnvoll sein zu prüfen, ob man den Protokollmechanismus des betroffenen Informationssystems entsprechend erweitern lassen kann, um diese Information im Protokoll verfügbar zu machen.

Im Zuge der praktischen Implementierung einer Lösungsarchitektur zur zeitnahen Nutzung von Process Mining Techniken, ist dieses Problem der unvollständigen Verbindungen zwischen Ereignissen und Prozessinstanzen aufgetreten. In Kapitel 4 wird das zu lösende Thema und der gewählte Ansatz zur Sicherstellung vollständiger Verbindungen im Detail diskutiert.

2.6.7 Präsentation von Prozessmodellen

Die Visualisierung der Ergebnisse des Process Minings sind der letzte wichtige Schritt zum erfolgreichen Abschluss der Durchführung des Process Minings. Den Entscheidungsträgern muss ein verständliches und aussagekräftiges Ergebnis präsentiert werden, mit dem diese neues Verständnis über die ablaufenden Geschäftsprozesse und eine Grundlage für weitere Entscheidungen erhalten können. Dies erfordert es auch, nicht-triviale Management-Informationen so zu modellieren, dass diese für den jeweiligen Empfänger übersichtlich und vor allem verständlich sind. Ein Terminus welcher in diesem Kontext oftmals verwendet wird ist Management-Cockpit, um der Bedeutung der Ergebnisvisualisierung mehr Gewichtung zu verleihen. Zur Präsentation von aus Ereignisprotokollen abgeleiteten Prozessmodellen gibt es einige – auch kommerzielle – Ansätze (ARIS PPM – ARIS Process Performance Monitor [26]).

Zusätzliche bekannte Herausforderungen (siehe [33]) im Zuge des Process Minings sowie weiterführende kontext-spezifische Herausforderungen sind im Laufe der Implementierung bzw. Umsetzung natürlich entsprechend zu berücksichtigen, werden jedoch in dieser Arbeit nicht vertiefend behandelt.

Verschiedene Mining Algorithmen können mit einigen der oben diskutierten Herausforderungen im Zuge des Process Minings besser umgehen als andere, d.h. man kann von einem starken Zusammenhang, zwischen Mining Algorithmus und den Herausforderungen, welche ausreichend gut behandelt werden können, ausgehen. Eine entsprechende Ausarbeitung zu dieser Thematik, welche für den Entscheidungsprozess hinsichtlich des zu verwendenden Algorithmus ausschlaggebend sein kann, kann man in Process Mining: A Research Agenda [33] nachlesen.

2.7 Process Mining Framework (ProM)

In den letzten Jahren wurden eine Vielzahl an Process Mining Algorithmen und Methoden entwickelt, aus denen einige Software Pakete entstanden sind. Im Besonderen wurde das Thema Process Mining durch die Technische Universität Eindhoven forciert und die Entwicklung von entsprechenden Tools und Frameworks durchgeführt. Ergebnisse dieser Arbeiten waren Tools wie EMiT [17], Thumb [54] und MISON [32], welche Mining und Analyse Algorithmen aus unterschiedlichen Perspektiven realisieren. Eine Integration dieser Entwicklungen und Ideen erfolgte im Zuge der Umsetzung des Open-Source Process Mining Frameworks ProM [40]. Die Idee des ProM Frameworks ist es, die Erkenntnisse der erwähnten Tools sowie deren unterschiedliche perspektivische Ansätze in einer Umgebung zu vereinen. Die geschaffene Umgebung kann mittels PlugIns, wobei diese unterschiedliche Probleme wie Import, Mining oder Analyse behandeln, individuell erweitert werden. D.h. es ist ohne Probleme möglich, eine spezifische Logik in

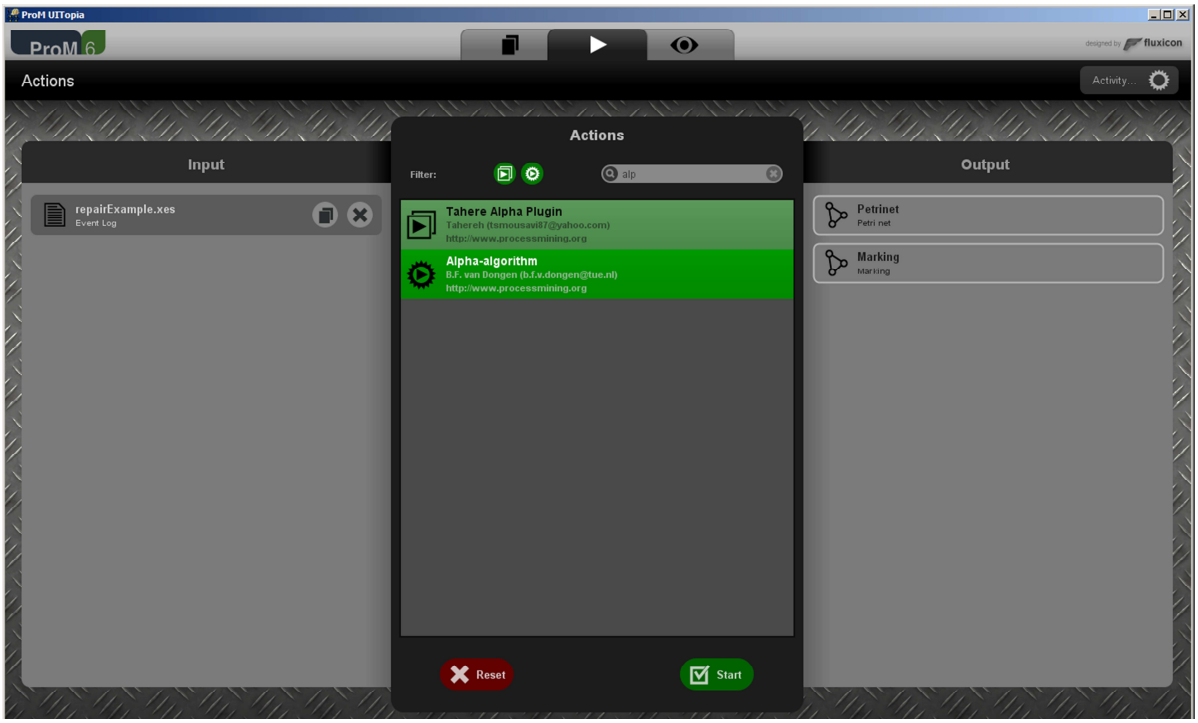

Abbildung 2.7: ProM 6 Framework - Mögliche Aktionen auf Ereignisprotokoll aus dem Objektpool

das Framework hinzuzufügen, ohne den Basiscode des Frameworks neu kompilieren zu müssen. Die aktuellste Version von ProM sowie viele nützliche Anleitungen, Beispieldatensätze und Hilfestellungen sind auf www.processmining.org verfügbar.

Die in das ProM Framework integrierbaren PlugIns können thematisch klassifiziert werden. Seit der Version 6 des ProM Framework gibt es, aus technischer Sicht, zwischen den in den vorhergehenden Versionen unterschiedlich klassifizierten PlugIn-Ansätzen keine Unterscheidung mehr. Konzeptionell spricht man nur noch von einem generischen PlugIn. Mit dieser Veränderung wurde auch das Konzept eines Objekt-Pools im ProM Framework realisiert. In diesem Objekt-Pool verwaltet das ProM Framework Objekte, welche aktuell in Verwendung sind oder waren (z.B.: Ereignisprotokoll, Modelle in Form eines Petri-Netzes, etc.). ProM Plug-Ins können Objekte aus dem internen Pool als Input Objekte verwenden und als Ergebnis der PlugIn-Ausführung wieder selbst Objekte für den Objekt-Pool produzieren. In manchen Sonderfällen macht es jedoch Sinn, dass ein PlugIn kein Objekt des Pools als Eingabeobjekt (z.B.: Import) verwendet oder auch gar kein Ergebnisobjekt (z.B.: Export) für den Pool produziert. Aus diesem Grund sollte die folgende Klassifizierung, welche technisch nicht mehr zwingend berücksich-

tigt werden muss, einen Überblick (aus inhaltlicher Sicht) über die unterschiedlichen im ProM Framework integrierbaren PlugIn Varianten geben.

2.7.1 Import PlugIns

Es wird eine Schnittstelle zur Verfügung gestellt, welche genutzt werden kann, um externe Ereignisprotokolle oder Referenzmodelle in den Objektpool zu laden. Diese Plug-In‘s können in die vorhandene Schaltfläche ’Ereignisprotokoll importieren’ eingegliedert werden. Wenn nun ein Ereignisprotokoll des dem PlugIn zugeordneten Format importiert werden soll, verwendet das ProM Framework das PlugIn zur Übersetzung in das interne Format. Hierzu wird dem zuständigen Import-PlugIn das zu importierende Ereignisprotokoll zur Verabeitung übergeben und das Ergebnis der Übersetzung in den Objekt-Pool geladen. Zusätzlich ist es möglich ein generisches PlugIn zu entwickeln, welche z.B. Ereignisprotokolldaten direkt aus einer relationalen Datenbank in das System laden und das Ergebnis – nach erfolgreicher Konvertierung - im Objekt-Pool ablegt.

2.7.2 Mining PlugIns

Das sind PlugIns, welche aufgrund eines vorhandenen Ereignisprotokolls ein entsprechendes Modell, z.B.: in Form eines Petri-Netzes, ableiten. Beispielweise wird hierfür ein importiertes Ereignisprotokoll aus dem Objekt-Pool als Eingabe verwendet, zur Detailkonfiguration des jeweiligen Algorithmus können durch den Benutzer spezifische Parameter – wenn notwendig - im jeweiligen PlugIn Dialog definiert werden. Das Ergebnis solch eines PlugIns könnte z.B.: ein soziales Netz sein, welches zur Detailanalyse verwendet werden kann. Dieses Ergebnis wird nach Durchführung der Mining Logik wieder im Objekt-Pool des Frameworks als Objekt abgelegt.

2.7.3 Conversion PlugIns

Es wird Funktionalität zur Verfügung gestellt, um eine Konvertierung in ein anderes Format vorzunehmen. Beispielweise wird ein abgeleitetes Prozessmodell in eine andere Darstellungsform übergeleitet.

2.7.4 Analysis PlugIns

PlugIns, welche aufgrund des Ergebnisses eines Mining PlugIns entsprechende Analysen durchführen. Hierzu werden vorhandene Objekte aus dem Objekt-Pool verwendet, z.B.: ein mittels eines Mining PlugIns abgeleitetes Prozessmodell und ein importiertes Referenzmodell werden auf Abweichungen analysiert.

2.7.5 Export PlugIns

PlugIns, welche die Ergebnisse der vorhergehenden PlugIns in externe Darstellungsformate exportieren. Dies kann bedeuten, dass z.B.: ein Petri-Netz für ein Folgesystem aufzubereiten ist

oder das konvertierte und modifizierte Ereignisprotokoll in ein bestimmtes Format abzuspeichern ist. Es werden ein oder mehrere Objekte des Objekt-Pools als Eingabe verwendet, Ergebnis der Ausführung des PlugIns kann, aber muss nicht zwingend, ein neues Objekt für den internen Pool sein.

Um die Ereignisprotokolle, welche beispielsweise durch ein Import-PlugIn in das Framework geladen wurden, im internen Objekt-Pool abzubilden, verwendet ProM ein XML basiertes Datenformat. In den früheren Versionen von ProM wurde MXML [38] zur internen Darstellung verwendet. Mit dem Zusatztool ProM Import Framework [12] war es möglich PlugIns zu entwickeln, mit welchen die Überführung von externen Datenformaten in MXML in eine unabhängige Applikation zusammengefasst werden konnte. Diese Applikation war jedoch nicht direkt in ProM integriert, sondern konnte separat zur Konvertierung der Protokolldaten vor der Benutzung von ProM verwendet werden. Wie bereits erwähnt ist mit neuesten Versionen die Integration von Import PlugIns direkt in ProM kein Problem mehr. Seit der Version 6 des ProM Frameworks wurde das Framework um das Format eXtensible Event Stream (XES) [41] [11] [10] erweitert, welches als direkter Nachfolger des MXML Format gesehen wird. Der Aufbau sowie die Unterschiede zu MXML und die daraus resultierenden Charakteristiken werden in den folgenden Kapiteln im Detail erläutert.

KAPITEL 3

Process Mining in Echtzeit vs. Zeitnahe Nutzung von traditionellen Process Mining Techniken

3.1 Allgemein

Wie in den vorhergehenden Kapiteln beschrieben, können durch den Einsatz von Process Mining Techniken unterschiedliche Anwendungsszenarien, wie Discovery, Conformance, Refinement und Extension, auf Basis von vorhandenen Ereignisprotokollen umgesetzt werden. Für die beschriebenen Anwendungsszenarien ist es üblicherweise ausreichend, auf Grund eines historischen Standes der Ereignisprotokolldaten, die entsprechenden Mining- und Analyse-Algorithmen anzuwenden. Beispielsweise ist es in spezifischen Anwendungsfällen ausreichend die Protokolldaten des letzten Jahres für die Generierung eines IST-Prozessmodells zu verwenden. Solch eine Betrachtungsweise berücksichtigt folglich keine über diese zeitlich abgegrenzten Bereich herausgehenden Prozessinstanzen, egal ob abgeschlossen oder noch in Durchführung. Traditionelles Process Mining stellt daher einen Offline-Analyseansatz von Prozessdaten dar. Diese Vorgehensweise ist für die besprochenen Anwendungsfälle ausreichend. Jedoch können die Ideen und Techniken auch für weitere Anwendungsszenarien eingesetzt werden, für welche diese klassische Offline-Betrachtung nicht genügend ist. Es mag beispielsweise von Interesse sein, dass man bei Abweichung des Prozessablaufs von kritischen Prozessen informiert wird und somit ein schnelleres Eingreifen bzw. eine schnellere Reaktion für die Betroffenen ermöglicht. Dies erfordert, dass die Ereignissprotokolle in Echtzeit oder nahezu in Echtzeit für Mining- und Analysealgorithmen zur Verfügung stehen und verwendet werden können.

Werden Ereignisse einer laufenden Prozessinstanz analysiert, bewertet und daraus weitere Aktionen abgeleitet, spricht man Unterstützung durch Process Mining in Echzeit oder nahezu in Echtzeit. Entsprechende Anwendungsszenarien für Process Mining in Echtzeit, werden folgend

auf Basis vorhanderer Literatur [34] [27] [25] [2] [29] [3] [39] [36] im Detail vorgestellt und die sich aus diesen Durchführungsszenarien ergebendenen Herausforderungen diskutiert. In [34] wird ein Framework zur operativen Unterstützung besprochen, welches mögliche Lösungswege für einige der vorgestellten Thematiken aufzeigt und diese im Detail behandelt. Mit diesem Framework versuchen die Autoren zu zeigen, wie entsprechende Systeme an das vorhandene Process Mining Framework (ProM) angebunden werden können, um Process Mining als operative Unterstützung verwendbar zu machen.

Der zweite in weiterer Folge disktutierte Ansatz ist, die traditionellen Anwendungsszenarien in einer beschleunigten Art und Weise zur Verfügung zu stellen. Dieses Ziel wird in weiterem Verlauf dieser Diplomarbeit als zeitnahe Nutzung der traditionellen Process Mining Techniken bezeichnet und vertiefend in den anschließenden Kapiteln disktutiert. Für diesen Zweck ist es notwendig die Techniken der traditionellen Anwendungsszenarien zeitlich näher an die Ereignisprokolldaten zu binden. D.h. der zeitliche Zwischenraum zwischen Anwendung der traditionellen Ansätze und den davor notwendigen Schritten (Extraktion, Aufbereitung, Konvertierung, etc.) kann durch optimierte Techniken reduziert werden. Ziel solch einer zeitnahen Nutzung ist es, den Aufwand zwischen Erzeugung der Prozessdaten und der Nutzung in entsprechenden Process Mining Techniken zu verringern, Abhängig vom Einsatzkontext kann dieser zeitliche Zwischenraum stark variieren. Im Zuge dieser Arbeit wurde solch ein Ansatz der Beschleunigung von traditionellen Anwendungsszenarien im praktischen Umfeld umgesetzt.

In den folgenden Absätzen werden die zwei unterschiedlichen Ansätze a) Process Mining zur operativen Unterstützung und b) Zeitnahe Nutzung von traditionellen Process Mining Szenarien sowie die jeweilig interessanten Herausforderungen besprochen. In den anschließenden Kapiteln wird die im Zuge dieser Arbeit gewählte Lösungsarchitektur zur zeitnahen Nutzung von traditionellen Process Mining Anwendungsszenarien im Detail erklärt und die daraus gewonnenen Schlüsse zum Abschluss präsentiert.

3.2 Anwendungsszenarien von Process Mining in Echtzeit

Process Mining in Echzeit erfordert es, auf die Ereignisprotokolldaten laufend zuzugreifen und nicht wie beim traditionellen Ansatz einen extrahierten historischen Teilbereich der Prozessdaten zu betrachten. Die Anwendungsszenarien lassen sich nach [34] in Check, Predict und Recommend (Abbildung 3.1) klassifizieren. Zu beachten ist, dass sich die traditionellen Anwendungsszenarien Discover, Conformance, Refinement und Extension auf die Generierung von Prozessmodellen fokusieren. Process Mining in Echtzeit stellt den aktuell laufenden Prozess in den Vordergrund. In Tabelle 3.1 werden die Anwendungsszenarien und deren Relation zu den unterschiedlichen Ansätzen illustrativ dargestellt.

3.2.1 Check

Ziel ist es, aufgrund der Ereignisse laufender Prozessinstanzen Abweichung vom Soll-Modell zu erkennen. Hierzu ist es notwendig die laufende Prozessinstanz mit einem bestehenden Prozess-

Szenario	Historische-Protokolldaten	Echtzeit-Protokolldaten
Check	X	X
Predict		X
Recommend		X
Discovery	X	
Conformance	X	
Refinement	X	
Extension	X	

Tabelle 3.1: Übersicht der Relationen zwischen unterschiedlichen Ansätze und möglichen Anwendungsszenarien [34]

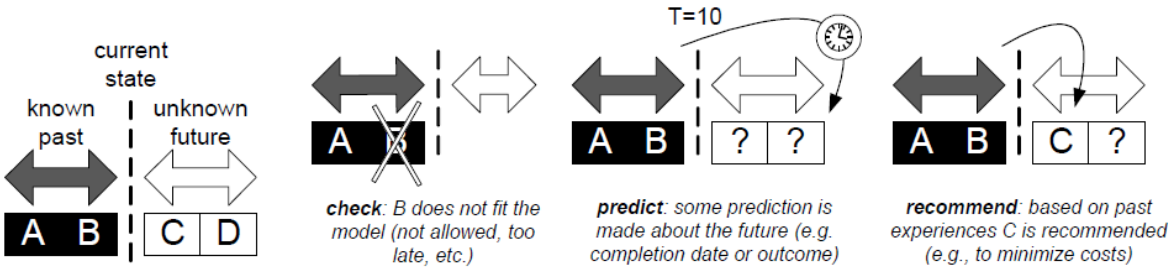

Abbildung 3.1: Process Mining in Echtzeit: Anwendungsklassen [34]

modell zu vergleichen. Sollte der Vergleich eine Abweichung aufzeigen, können entsprechende Aktionen gesetzt oder abgeleitet werden. Beispiele für Abweichungen wären z.B.: Ausnahmen in der Prozessausführung, nicht erlaubte Aktivitäten oder zeitliche Verfehlungen, wie die Überschreitung einer Deadline. In kritischen Prozessen, d.h. in Prozessen die eine hohe Priorität bzw. starke Auswirkungen auf andere Bereiche haben, können mit einer schnellen Erkennung von Abweichungen vom Soll-Prozess weitere Probleme vermieden werden.

3.2.2 Predict

Basierend auf dem bestehenden Prozessmodell bzw. Erfahrungen aus protokollierten Ausführungen, die durch klassisches Process Mining (Discovery) gewonnen wurden, können Vorhersagen für laufende Prozessinstanzen abgeleitet werden. Mögliche Vorhersagen wären beispielsweise die erwartete restliche Laufzeit der Prozessinstanz, die erwarteten weiteren Kosten des Prozesses oder auch das erwartete Ergebnis der Prozessausführung. Die gewonnen Aussagen können einen reinen Informationscharakter haben oder auch der Auslöser für notwendige Folgeentscheidungen sein. Ähnlich wie beim Anwendungsszenario Check wird ein rascheres Handeln bei nicht optimalen Prozessabläufen möglich. Übersteigt die erwartete Restlaufzeit das aktzeptable Maximum oder wird ein unakzeptables Prozessergebnis erwartet, können die Entscheidungsträger bzw. Prozessbeteiligten aufgrund der gewonnen Informationen entsprechende

Handlungen setzen.

3.2.3 Recommend

Aufgrund bestehender Erfahrungen (Soll- oder bereits abgeleitetes Ist-Prozessmodell) können den Prozessbeteiligten Vorschläge für weitere Aktionen im Rahmen des laufenden Prozesses unterbreitet werden. Solche Vorschläge sollen die Prozessdurchführung aus globaler Sicht optimieren. Die Idee von Recommend ist ähnlich der des Anwendungsszenarios Predict. Recommend versucht nicht nur vorauszusagen welche Ergebnisse zu erwarten sind, sondern versucht aufgrund aller möglichen Erwartungen die optimalste Möglichkeit zu wählen. Weiters können auch Vorschläge zur optimalen Verteilung von Ressourcen oder des besten Ausführungszeitpunkts mit solch einem Anwendungsszenario abgedeckt werden. Ziel dieser Anwendungen können Kosten- oder Durchlaufzeitoptimierung sowie die gezielte Auflösung von Ressourcen-Engpässen sein.

3.2.4 Herausforderungen

Will man Process Mining in Echtzeit verwenden, ergeben sich, zusätzlich zu den in Kapitel 2 beschriebenen Herausforderungen, weitere zu behandelnde Thematiken. Die folgenden Herausforderungen sind bei einer hierfür zu beachten:

- Wie können laufende Prozesse in Echtzeit verwendet werden?
- Welcher Grad an operativer Unterstützung ist zu erfüllen?

Ableiten laufender Prozesse

Betrachtungsgegenstand der Anwendungsszenarien Check, Predict sowie Recommend sind laufende Prozesse innerhalb eines Informationssystems. Um diese Szenarien durchführen zu können, ist es notwendig die laufenden Prozessinstanzen im Ereignisprotokoll zu identifizieren, anschließend zu extrahieren und entsprechenden Algorithmen zur Verfügung zu stellen. Eine Herausforderung besteht in der Erkennung dieser Prozessinstanzen. Im einfachsten Fall ist anzunehmen, dass alle Prozessinstanzen, welche den Soll-Endzustand des Prozesses noch nicht erreicht haben, als laufende Prozessinstanzen betrachtet werden dürfen. Diese Annahme ist jedoch nur für Informationssysteme gültig in denen alle Prozesse verlässlich immer den Endzustand erreichen. In der Praxis wird es oftmals Prozesse geben, welche aufgrund von Fehlern, Ausnahmen oder anderen Gründen den Endzustand nicht erreichen, jedoch als abgeschlossen betrachtet werden müssen. Um diese Thematik zu lösen sind zwei Möglichkeiten denkbar. Einerseits kann das Informationssystem den Prozess-Status (z.B.: Start, Abgeschlossen, Fehler, etc.) mitführen, was es ermöglicht zu beurteilen ob der Prozess evt. in einem anderen Prozessschritt beendet wurde. Dies würde das Problem von Prozessinstanzen, welche durch die Fehler- und Ausnahmebehandlung des Informationssystems nicht berücksichtigt wurden jedoch fehlerhaft beendet wurden, nicht lösen. Andererseits kann auch aufgrund einer zeitlichen abgegrenzten Betrachtung des Ereignisprotokolls (z.B.: alle Prozesse mit Start-Ereignissen aus der letzten Woche) die Wahrscheinlichkeit von Prozessen, welche fehlerhaft als laufend extrahiert werden, minimiert werden. Wichtig ist hierfür die Wahl eines adäquaten Betrachtungszeitraums, wobei

die durchschnittliche Durchlaufzeit der betroffenen Prozesse einen guten Anhaltspunkt bietet. Die beiden beschriebenen Ansätze können theoretisch kombiniert eingesetzt werden, um die fälschliche Identifikation von laufenden Prozessinstanzen zu minimieren und somit die Abfragequalität laufender Prozesse zu optimieren.

Sind Techniken und Regeln für die Extraktion von laufenden Prozessen vorhanden, ist es weiters notwendig entsprechende Techniken und Algorithmen zur Abbildung der Anwendungsszenarien zu implementieren. In [34] wird ein Framework zur operativen Unterstützung durch Process Mining vorgestellt und die für die Umsetzung getroffenen Annahmen sowie die daraus folgenden Herausforderungen diskutiert.

Grad der operativen Unterstützung

Ein grundlegende Überlegung muss sein, wie schnell die entsprechenden Process Mining Techniken auf die Aktualisierungen, d.h. auf das Hinzukommen von neuen Ereignissen, reagieren sollen. Diese Frage muss im Zuge der Anforderungsanalyse geklärt werden. Kritische Prozesse werden hierfür einen kürzeren Aktualisierungszyklus benötigen als unkritische. In weiterer Folge wird das Thema als Grad der operativen Unterstützung bezeichnet. Der Grad an operativer Unterstützung hat auf Entscheidungen hinsichtlich des Architekturdesigns sowie der Technik- bzw. Tool/Framework-Auswahl eine direkte Auswirkung. Auch weitere Herausforderungen (siehe folgend) stehen mit der Anforderung an den Grad der operativen Unterstützung direkt in Zusammenhang. Zu erwarten ist, dass ein höherer Grad höhere Kosten (in Umsetzung, Betrieb und Wartung) sowie Einschränkung in den Nutzungsmöglichkeiten (z.B.: manche Techniken oder Tools werden aufgrund dessen nicht anwendbar sein) verursacht.

3.3 Zeitnahe Nutzung von traditionellen Anwendungsszenarien

Viele der bestehenden traditionellen Process Mining Ansätze, welche die Anwendungsszenarien Discover, Conformance, Refinement oder Extension behandeln, gehen in ihren Annahmen von einem zeitlich abgegrenzten Ereignisprotokoll als Datenbasis aus. Aufgrunddessen wird in der vorhandenen Literatur kaum die Notwendigkeit einer Anbindung von Ereignisprotokollen in Nahezu-Echtzeit behandelt. Der Ablauf von traditionellen Anwendungsszenarien lässt sich üblicherweise in folgende Phasen gliedern:

- Extraktion der Ereignisprotokolldaten in ein externes Format
- Upload der Daten in ein Process Mining Framework
- Durchführung von Analyse- oder Mining-Algorithmen

Viele Implementierungen von traditionellen Methoden sehen auch keinen automatisierten Mechanismus zur Durchführung dieser einzelnen Phasen vor. Dieses beschriebene Vorgehen bedeutet, dass die Durchführung von Process Mining Techniken auf einem zeitlich klar abgegrenzten Ereignisprotokoll durchgeführt wird. Daraus folgt eine zeitliche Lücke zwischen Extraktion der Ereignisprotokolldaten und Durchführung der Mining- oder Analyse-Algorithmen.

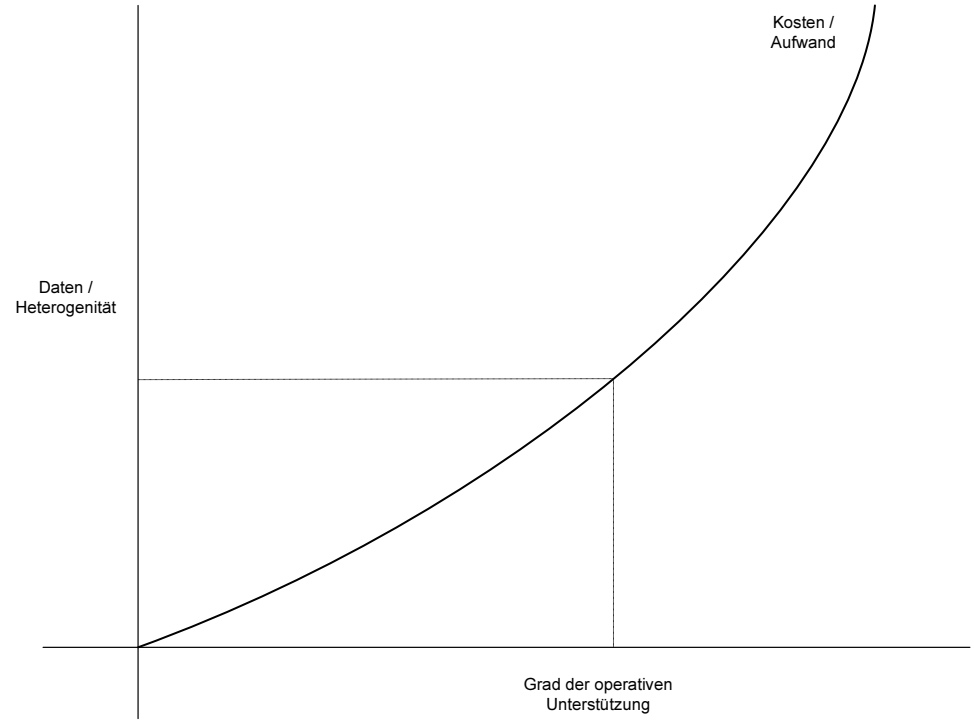

Abbildung 3.2: Process Mining in Echtzeit: Zusammenhang zwischen Grad der operativen Unterstützung und Datenmenge/heterogene Quellsysteme hinsichtlich Kosten und Aufwand

Alle Aktivitäten, welche nach der Extraktion aufgetreten sind, werden daher nicht berücksichtigt und haben auf die resultierenden Prozessmodelle keine Auswirkung. Sollten daher diese aufgetretenen Ereignisse zu einem anderen Prozessmodell führen (dieses erweitern oder verändern), muss der gesamte Prozess von Neuem gestartet werden.

Es mag jedoch in manchen Anwendungsfällen durchaus sinnvoll sein, traditionelle Anwendungsszenarien in einer beschleunigten Variante nutzbar zu machen. Ziel einer beschleunigten Verarbeitung kann es beispielsweise sein, vorhandene Zwischenschritte, wie die Konvertierung der Ereignisprotokolldaten in ein externes Darstellungsformat, durch entsprechende Mechanismen in der verarbeitenden Komponente (z.B.: Process Mining Technik oder Framework) zu erledigen. Dies ermöglich es die Prozessdaten folglich direkt nach Abruf nutzbar zu machen und stellt eine Nahezu-Echtzeitanbindung des Protokolls an die Process Mining Techniken dar. Ein weiteres Anwendungsbeispiel stellen Systeme mit flexiblen und variablen Prozessen oder dynamische Organisationen bzw. Organisationen mit einer dynamischen Umwelt dar. In solchen Fällen ist das System meist häufigen Anpassungen ausgesetzt, die folglich eine beschleunigte

Verarbeitung der traditionellen Process Mining Techniken erfordern.

Um eine beschleunigte Nutzung von traditionellen Anwendungsszenarien zu ermöglichen, muss der beschriebene Ablauf effizienter gestaltet werden. Eine Möglichkeit die Effizienz zu steigern, ist die Anzahl der benötigten Phasen zu reduzieren. Dies kann durch die Zusammenfassung mehrere Phasen zu einer Einzigen erreicht werden. Beispielsweise ist es durch eine direkte Anbindung der Ereignisprotokolldatenquelle an ein Process Mining Framework denkbar, die Extraktion aus dem Ereignisprotokoll sowie den Upload der Daten in das Framework, in einem automatisierten Schritt durchführen zu lassen. Solch eine Optimierung hat eine Reduktion der zeitlichen Lücke zwischen Extraktion und Mining zur Folge. Sollten die zu verwendenden Process Mining Algorithmen bekannt sein, ist es denkbar auch diesen Schritt automatisiert anzubinden. Jedoch verliert man durch solch eine Entscheidung, die Flexibilität andere Algorithmen oder Varianten zu verwenden, welche im Laufe der Zeit eventuell von Interesse sein könnten (z.B.: ergänzend eine organisatorische Betrachtung des Prozessmodells). Zusätzlich zu den organisatorischen Verbesserungen können auch durch die gezielte Technologieauswahl weitere Effizienzsteigerungen erreicht werden. Dies kann durch das Verwenden homogener Technologie (d.h. kompatible Schnittstellen) sowie durch die Auswahl von effizienten Datenspeichern erfolgen. Ziel der Technologieauswahl zur Effizienzsteigerung sollte die Reduktion von nicht zwingend benötigen Overhead-Kosten im ablaufenden Prozess sein. Im Zuge einer praktischen Implementierung wurden sowohl technische als auch organisatorische Maßnahmen zur Effizienzsteigerung und Beschleunigung von traditionellen Process Mining Anwendungsszenarien umgesetzt. Das resultierende Ergebnis wird in den folgenden Kapiteln im Detail diskutiert.

3.3.1 Herausforderungen

Zuzüglich zu den in Kapitel 2 beschriebenen Herausforderungen, ergeben sich bei einer zeitnahen Nutzung von traditionellen Anwendungsszenarien zusätzliche Thematiken. Folgend werden die zu erwartenden Herausforderungen und dafür mögliche Lösungsansätze erläutert, welche die nachstehenden Fragestellungen behandeln:

- Wie kann die Effizienz bei großen Datenmengen sichergestellt werden?
- Wie können Prozessdaten aus heterogenen Informationssystemen als Ausgangsbasis verwendet werden?
- Welche Konsequenzen hat die zeitnahe Nutzung von Process Mining Techniken für die Architektur?
- Wie zeitnah muss die Übernahme der Prozessdaten erfolgen? (siehe Grad der operativen Unterstützung)

Effizienz bei großen Datenmengen

Weitere Herausforderung ergeben sich, wenn große Datenmengen für die abgebildeteten Process Mining Lösungen verarbeitet werden müssen. Dies kann in zwei Szenarien von Interesse sein a)

wenn viele Prozesse mit hoher Frequenz ausgeführt werden (viele laufende Prozessinstanzen) und/oder b) falls das Prozessmodell ebenfalls häufig neu berechnet werden muss (z.B.: technische Einschränkung durch Benutzung vorhandener Techniken/Tools). Die Konsequenz daraus kann im Fall a) einerseits die Einschränkung der betroffenen Prozesse zur operativen Unterstützung sein oder im Fall b) die Verwendung anderer Techniken oder die Neuimplementierung analoger Lösungen. Beide Varianten bedeuten üblicherweise eine Reduktion der Flexibilität der Process Mining Lösung (z.B.: welche Techniken verwendet werden können). Um die Flexiblität innerhalb der Process Mining Lösung zu erhalten, kann, wenn es sich mit der Anforderung an die notwendige Lösung zu vereinbaren ist, der Grad an operativer Unterstützung entsprechend reduziert werden.

Ereignisprotokolle aus heterogenen Informationssystemen

Wie bereits in Kapitel 2 diskutiert, erschweren heterogene Quellsysteme die Durchführung des Process Minings. Aufgrund von unterschiedlichen Strukturen oder Granularitäten der Ereignisprotokolle sowie der Herausforderung zur Verknüpfung der Ereignisse zu einer Prozessinstanz ergibt sich, im Gegensatz zu der Verwendung eines homogenen Quellsystems, zusätzlicher Planungs-, Durchführungs- und Wartungsaufwand. Das Ziel der Gewährleistung einer operativen Unterstützung verschärft diese Thematik zusätzlich. Die bereits im vorhergehenden Kapitel vorgeschlagene Lösung, nämlich der Implementierung eines Data Warehouse (in diesem Fall ein Process Warehouse) mit entsprechenden Übernahmeprozessen, mindert die Problematik. Entsprechende Übernahmeprozesse in ein zentrales Protokoll müssten folglich auch in Nahezu-Echtzeit durchgeführt werden. Solche Prozeduren vermindern folglich den Grad an operativer Unterstützung, welcher bereitgestellt werden kann, um die Durchlaufzeit der Datenübernahme. Architekturen und Lösungswege für eine entsprechende Implementierung könnten sich an bestehenden Data Warehousing Lösungen orientieren, wie sie in [14] beschrieben sind.

Konsequenzen für die Architektur

Wie bereits erwähnt haben einerseits der Grad der operativen Unterstützung sowie die Designentscheidungen hinsichtlich der Effizienz bei großen Datenmengen eine direkte Auswirkung auf die Architektur der angestrebten Lösung sowie auf das anzuwendende Vorgehensmodell. Hinsichtlich der Kosten bzw. des notwendigen Aufwands der resultierenden Lösung ist mit hoher Wahrscheinlichkeit ein direkter Zusammenhang gegeben. In Abbildung 3.2 wird versucht diesen Zusammenhang zu illustrieren. Zu beachten ist, dass die Architektur an den Grad der operativen Unterstützung einer Lösung angepasst werden muss, d.h. eine Veränderung der Anforderung an den Grad der Unterstützung (im Besonderen, wenn dieser erhöht wird) hat auch mit großer Wahrscheinlichkeit ein (radikales) Redesign der Lösungsarchitektur zur Folge. Das in Kapitel 2 beschriebene abstrakte Vorgehensmodell ist auch für die Umsetzung einer operativen Unterstützung denkbar. Hierbei ist jedoch zu berücksichtigen den Datenfluß, (z.B.: nur Änderungen übernehmen) sowie die Rechenzeit (z.B.: Modell nur wenn wirklich notwendig aktualisieren), in der resultierenden Lösung so gering wie möglich zu halten. Hierzu ist das vorgeschlagene Vorgehensmodell sowie die Architektur den Anforderungen entsprechend anzupassen (erweitern oder modifizieren) und dieses angepasste Vorgehensmodell zu validieren.

KAPITEL 4

Prototyp zur Anbindung eines Apache Camel Ereignisprotokolls an das ProM Framework

4.1 Allgemein

In diesen Kapitel wird eine Implementierung zur zeitnahen Anbindung von traditionellen Process Mining Techniken an Ereignisprotokolldaten vorgestellt. Die Implementierung wurde im Kontext eines bestehenden unversitären Projekts ERPEL [45] entwickelt. Einige der archtiekturrelevanten Entscheidungen sind an diesem bestehenden Projekt orientiert getroffen worden. Ziel der Implementierung war es, die Ereignisprotokolldaten aus dem bestehenden System zeitnah als Input für Process Mining Algorithmen nutzen zu können. Da es im aktuellem Projekt noch keinen entsprechenden Protokollierungsmechanismus gab, war es notwendig, eine Erweiterung zu implementieren, welche die Ereignisdaten mit einem Protokollierungsmechanismus in eine geeignete Datenbasis protokolliert. Das bereits beschriebene Process Mining Framework ProM wurde aufgrund der Verfügbarkeit vieler unterschiedlicher Process Mining Algorithmen und der beschriebenen Möglichkeit zur Erweiterbarkeit als Umgebung zur Durchführung der Process Mining Algorithmen gewählt. Aufgrund dessen war es notwendig, eine Anbindung von ProM an das Ereignisprotokoll zu implementieren, um die Ereignisprotokolldaten in ProM für weitere Analysen direkt verwenden zu können. Einige der in der Diplomarbeit bereits diskutierten Herausforderungen zur zeitnahen Nutzung von Protokolldaten sowie zur Vorgehensweise bei der Durchführung von Process Mining, wurden in der gewählten Lösungsarchitektur berücksichtigt. Im folgenden Kapitel werden die Lösungsarchitektur sowie die Schlüsselelemente der Lösung vorgestellt.

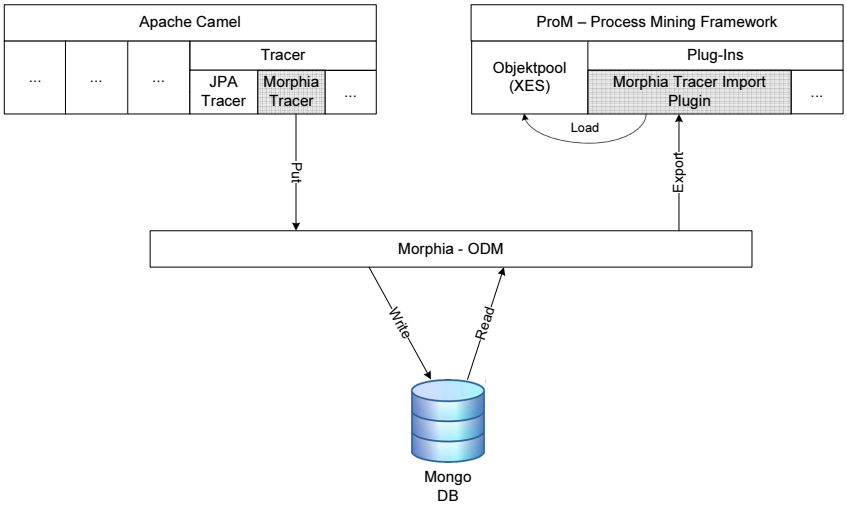

Abbildung 4.1: Praktische Umsetzung: Lösungsarchitektur

4.2 Lösungsarchitektur

Das Ziel einer zeitnahen Anbindung des Ereignisprotokolls in das Process Mining Framework ProM, erforderte den Entwurf einer entsprechenden Lösungsarchitektur (siehe 4.1). Für die Entwicklung dieser Architektur wurden einerseits notwendige Erweiterungen (Protokollierungs-, Import- und Konvertierungsmechanismus) als auch bereits bestehende und im Projektkontext verwendete Komponenten berücksichtigt. Die Technologieauswahl wurde auf die im bestehenden Projekt bereits verwendeten Technologien (z.B.: Datenbasis des Ereignisprotokolls, etc.) abgestimmt. Wie in der Lösungsarchitektur 4.1 dargestellt, umfasst die entwickelte Lösung folgende bereits vorhandene Komponenten und Technologien:

- **Apache Camel:** Integrations-Framework, welches im bestehenden Projektkontext verwendet wird, basierend auf Enterprise Integration Patterns
- **Apache Camel Tracer:** Implementierung eines Protokollierungsmechanismus für *Apache Camel*
- **MongoDB:** Dokumentbasierte Datenbank (bereits im bestehenden Projektkontext in Verwendung)
- **Morphia:** Object-Document Mapper (ODM) für die MongoDB

- **ProM - Process Mining Framework:** Unterschiedliche Process Mining Techniken zusammenfassende Umgebung (siehe Kapitel 4)

- **XES - eXtensible Event Stream:** XML-basierter Standard zur Darstellung von Ereignisprotokolldaten für Process Mining Techniken

Für die Umsetzung einer Lösung, welche die ausgeführten Prozesse im bestehenden Projektkontext protokolliert und diese in zeitnaher Form in ProM verfügbar macht, war es notwendig zwei zusätzliche Komponenten zu entwickeln. Zur Realisierung der Lösungsarchitektur mussten daher zwei weitere IT-Artefakte entworfen und umgesetzt werden:

- **Protokollierungsmechanismus - Morphia Camel Tracer:** Erweiterung des *Apache Camel Tracer*, um die Aktivitäten der Prozessinstanzen (im bestehenden Projektkontext ablaufend) in die gewählte Datenbank zu persistieren (Nutzung von Morphia als ODM)

- **Import- und Konvertierungsmechanismus - ProM Morphia Tracer Import PlugIn:** PlugIn um die protokollierten Prozessinstanzen aus dem Ereignisprotokoll in das XES-Format zu transformieren und folglich in das ProM Framework zu importieren. Das importierte Ereignisprotokoll steht somit für weitere Analyse- und Miningaufgaben zur Verfügung.

Im weiteren Verlauf dieser Arbeit, werden die im Zuge der Implementierung verwendeten Technologien *Apache Camel* und die dokumentorientierte Datenbank MongoDB mit der zugehörigen Persistence-API Morphia sowie das XML basierte Format XES zur Abbildung der Ereignisprotokolle im Detail vorgestellt. Aufbauend auf die verwendeten Technologien werden die im Zuge der Umsetzung entstandenen Artefakte a) Protokollierungsmechanismus: *Morphia Camel Tracer* und b) Import- und Konvertierungsmechanismus: *Morphia Camel Tracer ProM Import Plugin* diskutiert und die entsprechenden Implementierungsdetails erklärt.

4.3 Protokollierungsmechanismus

4.3.1 Einführung

Im Zuge der Diplomarbeit war es notwendig, einen Protokollierungsmechanismus zu entwickeln, welcher die Ereignisse von im bestehenden Projekt ablaufenden Prozessen protokolliert. Das bestehende Projekt nutzte das Integration-Framework *Apache Camel*. Da *Apache Camel* bereits eine grundlegende Implementierung eines Protokollierungsmechanismus (*Apache Camel Tracer*) zur Verfügung stellte, konnte dieser als Ausgangsbasis für unsere Implementierung verwendet werden. Als Datenspeicher für unser Ereignisprotokoll verwendeten wir die dokumentorientiete Datenbank Mongo-DB und für den Zugriff auf diese Datenbank das Morphia Framework. In den folgenden Punkten werden die verwendeten Technologien *Apache Camel*, MongoDB und Morphia sowie für die Implementierung relevante Aspekte dieser Technologien vorgestellt. Im Anschluss wird der entwickelte Protokollierungsmechanismus *Morphia Camel Tracer* diskutiert.

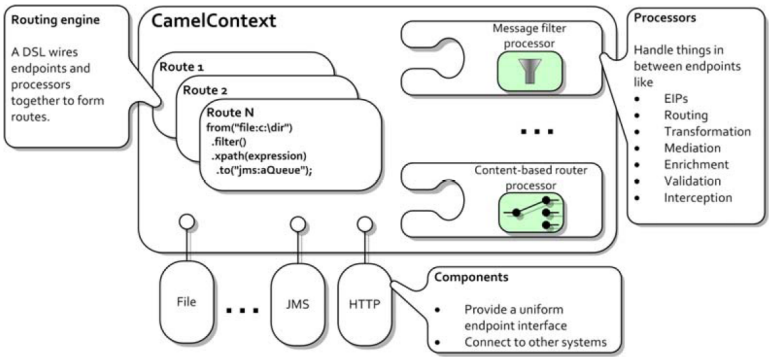

Abbildung 4.2: Apache Camel Architektur [16]

4.3.2 Apache Camel

Apache Camel [16] [19] [42] ist ein noch relativ junges (Anfang 2007) Integrations-Framework, bei welchem der Fokus auf die simplifizierte Einbindung von unterschiedlichen Applikationen unter Verwendung von Enterprise Integration Patterns (EIP) [15] gelegt wurde. Das Herzstück von *Apache Camel* ist die *Routing Engine* bzw. der *Routing Engine Builder*, welcher es ermöglicht eigene Routing Regeln zu definieren. Diese Regeln beschreiben von welchen Quellen Nachrichten angenommen werden sowie wie die Systematik zur Verarbeitung und Weiterleitung dieser eingehenden Nachrichten aussehen soll.

Ein grundlegendes Prinzip des EIP Frameworks stellt die Unabhängigkeit von den zu verarbeitenden Daten sowie den Kommunikationsprotokollen der einzelnen integrierten Systeme dar. Dies erleichtert die Einbindung unterschiedlichster Systeme in das Framework, da Überführungsprozesse der Daten der einzelnen Systeme in eine normalisierte Form nicht zwingend notwendig sind. *Apache Camel* versucht durch einen hohen Grad an Abstraktion, die Vermittlung und Interaktion zwischen unterschiedlichsten Systemen über eine einheitliche API zu ermöglichen, ohne auf die unterschiedlichen Protokolle und Datentypen der einzelnen Systeme bewusst Rücksicht nehmen zu müssen.

Die zentralen Bestandteile von *Apache Camel*, wie in der Architekturübersicht (Abbildung 4.2) dargestellt, sind einerseits der *CamelContext* – welcher als Container der gesamten Systemkonfiguration agiert - sowie *Components*, *Processors* und *Routes*. Die *Components* in *Camel* sind als Erweiterung zur Konnektivität zu anderen Systemen zu sehen und stellen eine einheitliche *Endpoint*-Schnittstelle zur Verfügung. Diese *Endpoints* können mittels *Routes*, d.h. definierte Routing Regeln, mit den *Processors* verdrahtet werden. Die *Processors* implementieren EIP Patterns [16] unter anderem mit der Aufgabe zur Verarbeitung und Transformation von gerouteten Nachrichten.

In den folgenden Abschnitten werden die zentralen Bestandteile der *Camel* Architektur sowie das grundlegende Nachrichten-Modell von *Camel* genauer betrachtet.

Routes

Wie bereits eingangs erwähnt, stellen *Routes* ein gundlegendes Konzept des *Apache Camel EIP Frameworks* dar. *Routes* ermöglichen es, die konkrete Verarbeitungsequenz eingehender Nachrichten in *Camel* zu konfigurieren. *Apache Camel Routes* enthalten mehrere *Camel Processors*, welche das Ergebnis des vorhergehenden *Processors* als Eingangsnachricht erhalten, entsprechend der jeweiligen Business-Logik des *Processors* verarbeiten und an den folgenden *Processor* weitergeben. Der erste *Processor* in der Kette ist Abnehmer eines *Endpoints* und setzt die Verarbeitung dieser Eingangsnachricht in der durch die *Route* definierten Prozesskette in Gange. Jede *Camel Route* kann nur eine einzige Quelle für Eingangs-*Messages* haben, d.h. eine *Route* ist immer an einen Eingangs-*Endpoint* gebunden. *Camel Routes* ermöglichen eine strikte Trennung von Client und Server bzw. von Erzeuger und Abnehmer innerhalb einer Messaging Applikationen. *Routes* erlauben es auf flexible Art zusätzliche Verarbeitungsschritte in einen bestehenden Prozess einzugliedern und bieten die Möglichkeit zur unabhängigen Entwicklung von Client und Server der Applikation. Auch ein dynamischer Entscheidungmechanismus, beispielsweise um zu entscheiden welcher Server von den Clients aufgerufen werden soll, ist mit diesem Konzept umsetzbar.

Apache Camel erlaubt es *Routes* auf unterschiedliche Art und Weise zu definieren. Einerseits bietet *Apache Camel* eine Domain-specific Language (DSL) an, mit welcher *Routes* definiert werden können. Diese DSL ist eine Java API, welche Methoden, benannt nach EIP Begriffen, zur Definition von *Routes* anbietet (siehe Beispiel 4.1). Die im Beispiel definierte *Route* konsumiert Dateien von einem File-*Endpoint* und leitet diese an ein Filter EIP weiter. Dieser Filter verwendet einen XPath-Ausdruck um zu entscheiden, ob die *Message* an den JMS-*Endpoint* (Java Message Service) weitergeleitet oder unverarbeitet verworfen wird. Zusätzlich werden noch weitere bestehende DSLs unterstützt, wie beispielsweise die XML-basierte Variante *Spring DSL*. Listing 4.2) zeigt die Definition der selben *Camel Route* als im vorhergehenden Beispiel mit *Spring DSL*.

```
1 from("file:data/inbox")
2 .filter().xpath("/order[not(@test)]")
```

Listing 4.1: Definition einer Route mit Camel DSL (siehe [16])

```
1 <route>
2     <from uri="file:data/inbox"/>
3     <filter>
4         <xpath>/order[not(@test)]</xpath>
5         <to uri="jms:queue:order"/>
6     </filter>
7 </route>
```

Listing 4.2: Definition einer Route mit Spring DSL (siehe [16])

An jede in *Apache Camel* definierte *Route* wird ein eindeutiger Bezeichner vergeben, welcher zum Starten oder Stoppen dieser *Route* in *Camel* sowie für weiterführende Überwachungs- und Protokollierungszwecke verwendet werden kann.

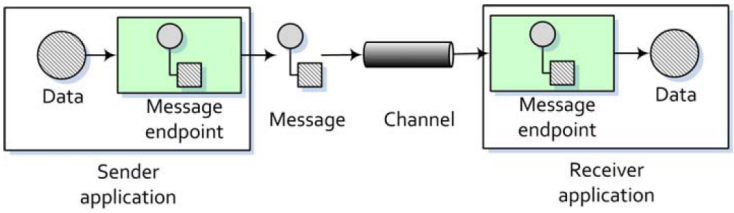

Abbildung 4.3: EIP - Channel Pattern [16]

Components

Components stellen die Erweiterungspunkte hinsichtlich der Konnektivität zu anderen Systemen dar. Mittels dieser Komponenten ist es für *Apache Camel* möglich, Nachrichten über ein bestimmtes Protokoll oder Middleware zu versenden und zu empfangen. *Camel* stellt eine große Anzahl von Komponenten zur Verfügung. Beispiele für solche Komponenten sind die JMS-Komponente zur Verbindung von jeglichen JMS-Providern oder die File-Komponente welche zum Lesen und Schreiben von Dateien genutzt werden kann. Eine *Component* wird von *Apache Camel* zum Erzeugen von *Camel Endpoints* verwendet, welche in weiterer Folge zum Empfangen und Senden von Nachrichten genutzt werden können. *Components* werden in *Apache Camel* mittels eines eindeutigen Namens identifiziert, der als Schema bezeichnet wird und für die Erzeugung von entsprechenden *Endpoints* via *Camel* berücksichtigt werden muss. Die entsprechende Konfiguration kann via einer Konfigurationsdatei (z.B.: *Spring XML*) oder durch Nutzung des *CamelContext*-Objekts (Listing 4.3) direkt im Applikationscode erfolgen. Die zweite Variante wird in Normalfall bei Komponenten mit komplexer Konfiguration eingesetzt, um die Wartbarkeit des Systems sowie der Konfigurationsdateien zu erhöhen.

```
1 ConnectionFactory connectionFactory =
2       new ActiveMQConnectionFactory("tcp://myServer:61616");
3 Component myComponent =
4       JmsComponent.jmsComponentAutoAcknowledge(connectionFactory);
5
6 context.addComponent("myComponent", myComponent);
```

Listing 4.3: Konfiguration einer Component via Camel Context [19]

Endpoints

Wie bereits erwähnt, ist das *Apache Camel* Konzept der *Components* eng mit dem der *Endpoints* verknüpft. Stellen *Components* die Erweiterungspunkte hinsichtlich der Konnektivität dar, können die *Endpoints* als konkrete Instanzen dieser Erweiterungspunkte gesehen werden. *Camel Endpoints* ermöglichen an das angebundene System (mittels des *Endpoints*) Nachrichten zu senden oder von diesem Nachrichten zu empfangen. Ein *Camel Endpoint* stellt, wie in Abbildung 4.3 abgebildet, den Channel des Channel Patterns dar [16].

Endpoints werden in *Camel* als Uniform Resource Identifier (URI) repräsentiert, wobei sich dieser aus drei Teilen zusammensetzt: a) Schema, b) Kontextpfad und c) Optionen (siehe Beispiel 4.4). Das Schema der URI orientiert sich an dem eindeutigen Namen der entsprechenden Komponente. Im abgebildeten Beispiel ist das Schema *pop3* - der eindeutige Name der POP3-Komponente. Der Kontextpfad stellt die konkrete Verbindungsinformation (im Beispiel die eMail-Adresse) zur jeweiligen Schnittstelle dar. Im dritten Teil können (dieser Teil kann als optional betrachtet werden) zusätzliche Optionen für den *Endpoint* angegeben werden. Im abgebildeten Beispiel ist das Passwort für die Authentifikation am POP3-Server als zusätzliche Option angegeben. Zum Erzeugen des *Endpoints* wird die zugehörige *Component* von *Apache Camel* als Factory verwendet.

```
1 pop3://thomas.neuboeck@any.mailhost.com?password=anyPassword"
```

Listing 4.4: URI eines POP3 Endpoints

Das Versenden und Empfangen von Nachrichten erfolgt durch die Verwendung von *Producer*- und *Consumer*-Instanzen, welche vom jeweiligen *Endpoint* erstellt werden. In *Apache Camel* konsumiert ein *Consumer* Nachrichten vom den *Endpoint* zugrunde liegenden Channel, ein *Producer* fügt neue Nachrichten zu diesen Channel hinzu. Dies bedeutet die Instanzen verwenden den jeweiligen *Endpoint*, wie in Abbildung 4.3 dargestellt, als Channel für die Kommunikation untereinander. *Apache Camel* unterstützt für die Umsetzung des Consumers Teil das *Event-driven Consumer Pattern*. Für diese Pattern-Realisierung wird ein *Camel Processor* zur Verfügung gestellt, welcher beim Eintreffen einer neuen Nachricht sofort ausgeführt wird. Ebenfalls unterstützt *Apache Camel* den Einsatz des *Polling Consumer Patterns*. Bei diesem fragt der *Consumer* selbständig nach, ob neue zu verarbeitende Nachrichten existieren. Ein Producer übernimmt hierbei die Aufgabe, neue Nachrichten über den *Channel* (der jeweilige *Endpoint*) den *Consumern* zur Verfügung zu stellen, welche von diesen regelmäßig abgeholt werden.

Processors

Ein weiteres bedeutendes Konzept von *Apache Camel* sind *Processors*, welche die Aufgabe zur Verarbeitung eingehender *Messages* haben. Solche Verarbeitungsschritte sind beispielsweise Transformationen der eingehenden *Messages* oder Manipulation dieser. Wie bereits erwähnt, erhält ein *Processor* immer das Ergebnis des vorhergehenden *Processors* und gibt nach erfolgreicher Verarbeitung sein Ergebnis an den nachfolgenden *Processor* weiter. Ein *Camel Processor* kann auch Routing-Aufgaben übernehmen, indem durch den jeweiligen *Processor* entschieden wird an welchen Consumer die Nachricht weiterversendet wird (Implementierung des Routing Pattern). Einige der bereits in *Apache Camel* vorhandenen *Processors* implementieren EIPs (*ChoiceProcessor* oder *FilterProcessor*), jedoch kann die Erweiterung um individuelle *Processor*-Implementierungen relativ einfach erfolgen. Zur Entwicklung eines *Processors* in *Apache Camel* ist es notwendig das Interface *Processor* (siehe Listing 4.5) zu implementieren. Ein *Processor* muss lediglich eine Methode *process* implementieren, welche den *Exchange* vom vorhergehenden *Processor* der *Route* übermittelt bekommt. Auf diesem *Exchange*-Objekt hat der *Processor* die implementierte Business Logik auszuführen.

```
1 package org.apache.camel;
```

```
2
3 public interface Processor {
4     void process(Exchange exchange) throws Exception;
5 }
```

Listing 4.5: Apache Camel Processor Interface

CamelContext

Der *CamelContext* ist wie bereits erwähnt, als zentraler Konfigurationscontainer des *Apache Camel EIP Frameworks* zu sehen. Eine Instanz, welche ein spezifisches Integrationsszenario abbildet, enthält einerseits alle in der Applikation genutzten *Components*, die aufgrund dieser erstellten *Endpoints* sowie alle *Routes*, welche für die jeweilige Applikation definiert wurden. Weiters stellt eine Instanz des *CamelContexts* geladene Typ-Konverter – welche vom *Camel* Mechanismus zur Umwandlung von Datentypen genutzt werden – sowie geladene Datenformate und Expression Languages zur Verfügung. Zusätzlich enthält der Apache *CamelContext* eine Registry – im Standardfall ist dies eine JNDI Registry, bei der Verwendung von *Spring* der *Spring ApplicationContext* – um Java Beans (*Processors* oder *Endpoints*) zu verwalten.

Nachrichtenmodell

Das in *Camel* verwendete Nachrichtenmodell umfasst aus zwei grundlegenden Konzepten a) *Message* und b) *Exchange*. Eine *Camel Message* stellt die fundamentale Einheit, welche innerhalb der *Camel Routen* transportiert wird dar. Ein *Camel Exchange* bildet die Kommunikationseinheit zwischen *Processors* ab, welcher aus der Anfrage In-*Message* und der Antwort Out-*Message* besteht. Für die Entwicklung von Applikationen ist primär der Zugriff auf das *Exchange*-Objekt sowie das *Message*-Objekt interessant. Folgend werden die beiden zentralen Elemente des *Apache Camel Nachrichtenmodells* beschrieben.

Message

Eine *Message* in *Camel* stellt die Kommunikationseinheit zwischen den einzelnen Systemen dar. Sie besteht, siehe Abbildung 4.4, aus drei Bestandteilen: a) Header, b) Body und c) einem optionalen Anhang. Eine Nachricht wird innerhalb von *Camel* mittels eines eindeutigen Bezeichners beschrieben. Dieser eindeutige Bezeichner wird vom Ersteller der Nachricht zur Verfügung gestellt und ist protokollabhängig. Stellt das Protokoll diesen Identifikator nicht zur Verfügung verwendet *Camel* einen internen UID Generator.

Die Headerinformationen einer *Message* sind als ein Key/Value-Paar umgesetzt. Der eindeutige Name ist als case-insensitive *java.lang.String* abgebildet. *Apache Camel* hat für den Inhalt der *Message* keine Typrestriktionen vorgesehen, folglich sind die Werte der einzelnen Headereinträge als *java.lang.Object* abgebildet. Im Body einer *Message* kann jeglicher Inhalt abgelegt werden. Dies bedeutet jedoch auch, dass der Entwickler sicherstellen muss, dass der Nachrichtenempfänger den Inhalt der Nachricht auch entsprechend verstehen und folglich weiterverarbeiten kann.

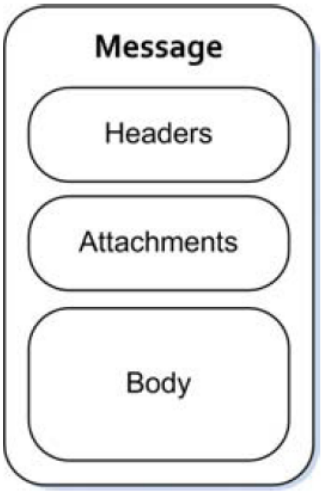

Abbildung 4.4: Nachrichtenmodell: Camel Message [16]

Exchange

Unter *Exchange* versteht man die Einheit, die im Zuge des Routing Prozesses zwischen den einzelnen *Camel Processors* ausgetauscht wird. Er ist ein Container-Objekt, welches unter anderem die für die Verabeitung relevanten *Messages* kapselt (siehe Abbildung 4.5). Ein *Camel Exchange* unterstützt unterschiedliche Message Exchange Patterns (MEP): a) InOnly: Nachricht ohne Antwort sowie b) InOut: Nachricht mit Antwort. Der Container enthält die Eingangsnachricht und optional die Ausgangsnachricht, abhängig vom genutzten MEP, sowie eine eindeutige *ExchangeID*. Diese ID wird entweder von *Apache Camel* automatisch generiert oder durch den Entwickler innerhalb der Applikation erstellt und gesetzt. Des Weiteren enthält ein *Exchange* Informationen zum verwendeten MEP sowie zu Exceptions, welche im Zuge der Verarbeitung über die gesamte *Route* hinweg aufgetreten sind. Zusätzlich kann ein *Exchange* auch weitere spezifische Eigenschaften enthalten, welche über den ganzen Verlauf der *Route* bestehen bleiben. Am Beispiel (Listing 4.6) wird der Zugriff auf eine *In-Message* des *Exchange*-Objekts mittels der entsprechenden Eigenschaften des Objekts gezeigt.

```
1 public void process(Exchange exchange) {
2     // get the in request message:
3     Message msg = exchange.getIn();
4
5     System.out.println("Processing: " + msg.getBody(String.class));
6 }
```

Listing 4.6: Apache Camel Nachrichtenmodell - Zugriff auf Request-Message via Exchange-Objekt

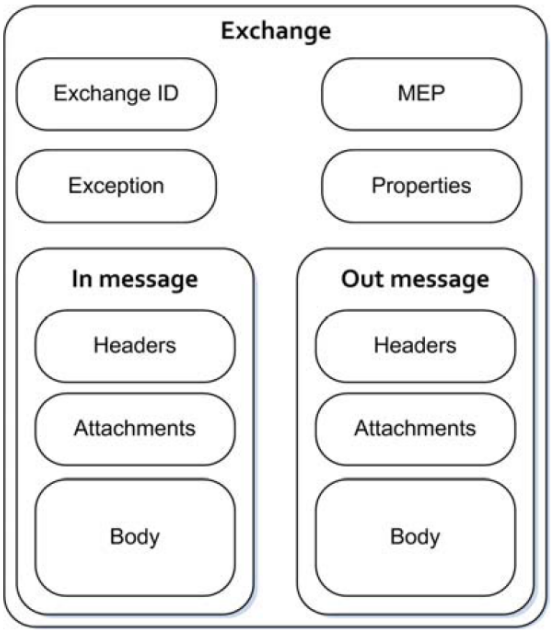

Abbildung 4.5: Nachrichtenmodell: Camel Exchange [16]

Camel Tracer

Apache Camel stellt, wie Eingangs erwähnt, einen eigenen Protokollierungsmechanismus - genannt *Camel Tracer* - zur Verfügung. Dieser Mechanismus ermöglicht es, im Zuge der Ausführung einzelner *Camel Routes*, relevante Informationen über die aufgetretenen Ereignisse (z.B.: Übergänge zwischen einzelnen *Processors*) zu protokollieren. Der *Apache Camel Tracer* ist mittels Interceptor-Strategie (Abbildung 4.6) umgesetzt. Dies bedeutet, dass der *Camel Tracer* auf den Eintritt eines Ereignisses, im Zuge der *Routes*-Durchführung, reagiert. Hierzu erhält der *Camel Tracer* vom Framework die entsprechenden Informationen zu dem eingetretenen Ereignis (z.B.: das jeweilige *Exchange*-Objekt sowie den Konfigurationscontainer *CamelContext*) und führt die entsprechend implementierte Logik (z.B.: Ausgabe von Informationen auf der Standardausgabeschnittstelle) durch. Der *Apache Camel Tracer* berücksichtigt hierzu jeden Aufruf aller definierten Knoten (d.h. *Processors*) innerhalb der ausgeführten *Route*. Wie in der Abbildung 4.6 dargestellt, wird der *Apache Camel Tracer* bei jedem Aktionsübergang durch das Framework ausgelöst (z.B.: zwischen Ereignis A und B) und kann somit entsprechende Informationen im Zuge des Aktionsübergangs verarbeiten und protokollieren. *Apache Camel* stellt eine Standard-Implementierung (d.h. Ausgabe der Ereignisdaten in der Standardausgabe) dieses *Tracers* sowie eine spezifische JPA Implementierung zur Verfügung. Es ist jedoch möglich das *Camel* Framework durch eine eigene Implementierung zu erweitern, wobei hierzu die JPA Referenz-Implementierung als Best-Practice Vorlage herangezogen werden kann. Zur Erweiterung des *Apache Camel Tracers* stellt das Framework die Möglichkeit zur Verfügung, einen eigenen *TraceHandler* zu implementieren, um spezifische Anforderungen im jeweiligen System

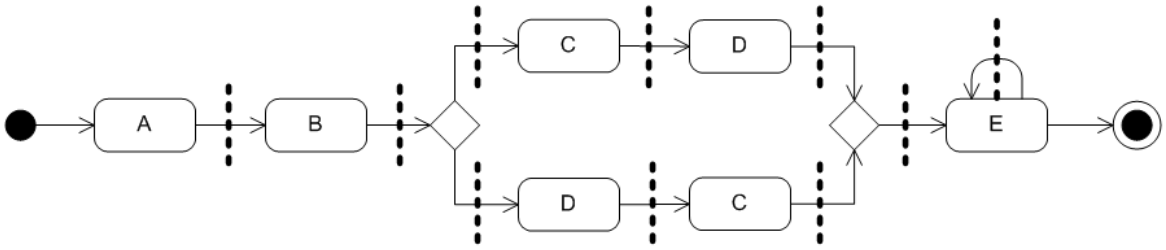

Abbildung 4.6: Apache Camel Tracer: Schematische Darstellung der Funktionsweise des Tracers

zu ergänzen.

Eine Instanz des *Apache Camel Tracers* kann mittels folgenden Optionen konfiguriert werden:

- **Formatter:** Instanz, welche das Ausgabeformat des entsprechenden Ereignis beschreibt
- **Enable:** Schalter zum Aktivieren oder Deaktiveren der Tracer-Funktionalität
- **TraceFilter:** Möglichkeit, um relevante *Exchanges* zu filtern
- **TraceExceptions:** Aktiviert das Protokollieren von Exceptions
- **TraceOutExchange:** Aktiviert das Protokollieren von Out-Messages, per Default werden nur In-Messages berücksichtigt
- **UseJPA:** Aktiviert die JPA (Java Persistence API) Protokollierung.
- **TraceHandler:** Definiert eine eigene Klasse, welche die Kontrolle/Verarbeitung bei Ereignisauftritt übernehmen soll

Will man das *Apache Camel* Framework um einen eigenen individuellen Protokollierungsmechanismus, welcher die im Zuge der Ausführung aufgetretenen Ereignissen verarbeitet und protokolliert, erweitern, muss, wie bereits erwähnt, ein entsprechender *TraceHandler* sowie optional eine eigene *TraceEventMessage* implementiert werden. Ein *TraceHandler* übernimmt die Aufgabe der Kontrolle bei Ereigniseintritt, um die entsprechende Protokollierungslogik anzuwenden. Konfiguriert man *Apache Camel* zur Nutzung einer eigenen Implementierung eines *TraceHandlers*, delegiert *Apache Camel* beim Ereigniseintritt die Verarbeitung der Ereignisdaten an diese Implementierung weiter. Ansonsten übergibt *Apache Camel* die notwendige Informationen an die Standard-Implementierung des *TracesHandlers*. Eine *TraceEventMessage* ist in der Standard-Implementierung des *Tracers* zwar nicht vorgesehen, ist jedoch sinnvoll, sobald die Ereignisdaten in einer strukturierten Form persistiert werden sollen (z.B.: bei der Verwendung

eines Object-Mapper zur Persistierung in eine Datenbank). Hierbei dient die *TraceEventMessage* zur Beschreibung des aufgetretenen Ereignisses, d.h. sie enthält alle für die Protokollierung notwendigen Eigenschaften beinhaltet jedoch keine eigene Business Logik. Eine entsprechende *TraceEventMessage* wird im Normalfall als Plain Old JavaObject (POJO) umgesetzt. Eine Erweiterung des *Apache Camel Frameworks* um einen eigenen *TraceHandler* sowie die relevanten Entwicklungsdetails werden im weiteren Verlauf dieser Arbeit noch ausführlich beschrieben.

4.3.3 MongoDB: Dokumentorientierte Datenbank

Um die vom *Apache Camel Tracer* erzeugten Ereignisprotokolldaten dauerhaft verfügbar zu halten, ist es notwendig, eine adäquate technologische Lösung zur Datenspeicherung zu wählen. Die Anforderung die Ereignisprotokolldaten für die Verwendung in Process Mining Techniken so echtzeitnah wie möglich verfügbar zu machen sowie im Gesamtprojektkontext eine homogene technologische Architektur zu erzielen, sollte bei dieser Auswahl berücksichtigt werden. Im Zuge der praktischen Umsetzung wurde die Entscheidung getroffen, keine relationale Datenbank (RDB) als Datenbasis für das Ereignisprotokoll zu verwenden sondern eine dokumentorientierte Datenbanklösung einzusetzen. Die Entscheidung zur Verwendung der plattformunabhängigen Open Source No-SQL Datenbank Mongo DB [4] [47] wurde zusätzliche durch den den Einsatz im bestehenden universitäten Projekt (welches durch den Protokollierungsmechanismus erweitert werden soll) begünstigt. Ziel war es, hinsichtlich der Erweiterung von ERPEL, eine homogene Technologieauswahl für die Architektur zu treffen. Weitere Argumente für den Einsatz der MongoDB werden in den folgenden Absätzen erläutert sowie die Grundkonzepte von der dokumentbasierten Datenbank MongoDB erklärt.

Allgemein

MongoDB ist eine dokumentorientierte plattformunabhängige schema-free Datenbanklösung, welche in C++ als Open Source Projekt entwickelt wurde. Der Name Mongo lässt sich durch den englischen Begriff “humongous” ableiten, was übersetzt riesig/gigantisch bedeutet. Die grundlegenden Konzepte der dokumentorientierten Datenbank MongoDB sind:

- Basiseinheit zur Datenspeicherung ist ein Dokument
- Dokumente müssen keinem fix definierten Schema folgen
- Dokumente werden in Collections zusammengefasst
- Dokumente haben einen eindeutigen Schlüssel innerhalb der jeweiligen Collection
- MongoDB unterstützt keine Transaktionen
- Daten werden im Hauptspeicher gehalten und durch einen Hintergrund-Thread auf die Festplatte geschrieben

Ein MongoDB Server kann aus mehreren unabhängigen Datenbankinstanzen bestehen. Ein Datenbank selbst setzt sich aus mehreren unterschiedlichen Collections von Dokumenten zusammen. Weiters enthält jede Datenbankinstanz eine eigene Berichtigungsdefinition, in welcher

für die Benutzer geltende Zugriffsregeln definiert und verwaltet werden können. Die MongoDB wird um weitere Funktionaltitäten wie Indexing sowie Replikation erweitert, was die MongoDB zu einem skalierbaren und robusten Datenspeicher macht. In Folge werden die zentralen Elemente a) Dokumente und b) Collections der MongoDB sowie Vor- und Nachteile in der Verwendung der dokumentorientierten Datenbank erläutert. Für den Zugriff auf die MongoDB stehen eine eigene Abfragesprache (basierend auf JavaScript) sowie eine Java API zur Verfügung, welche in etwaigen eigenen JAVA Applikationen verwendet werden kann. Die notwendigen Fakten zu dieser API werden im Zuge der Erläuterung der praktischen Implementierung erklärt.

Dokument

Daten werden in der MongoDB in Form von Dokumenten gespeichert. Diese Dokumente müssen, wie bereits erwähnt, im Gegensatz zu einzelnen Relationen in einer RDB, keinem fix definierten Schema folgen. Dokumente werden in der MongoDB als BSON [44] Dokumente, eine binäre Variante der JavaScript Object Notation (JSON) [6], dargestellt. JSON ist ein programmiersprachen-unabhängiges, kompaktes Datenformat, welches zum Datenaustausch zwischen Applikationen eingesetzt wird. Grundidee von JSON ist es ein sowohl für den Mensch, als auch für die Maschine lesbares Format bereitzustellen. JSON wird oft anstatt XML in Bereichen eingesetzt, in denen vorhandene Ressourcen optimal und sparsam genutzt werden. Mittels JSON können Informationen in unterschiedlichen Formaten, auch verschachtelt, beschrieben werden. Beispiele für Datentypen sind bool'sche Werte, Zahlenwerte oder Zeichenketten, sowie Arrays oder Objekte. Informationen in JSON werden immer, unabhängig vom Datentyp, als Key-Value Paar dargestellt. Binary JavaScript Object Notation (BSON) stellt eine binäre-kodierte Serialisierung von JSON dar. BSON erlaubt die Darstellung der selben Datentypen wie mittels JSON. Zusätzlich stellt BSON Erweiterungen zur Verfügung, beispielsweise zusätzliche Datentypen wie *Date*. BSON wurde basierend auf drei Grundsätzen entworfen: a) Lightweight - minimaler Overhead, b) Traversable - Einfache Verabeitung und Durchlauf der Daten und c) Efficient - Schnelle Kodierung bzw. Dekodierung der Daten. Diese Grundsätze ermöglichen eine hohe Performance beim Zugriff, bei der Verarbeitung und beim Austausch der BSON Dokumente. In Listing 4.7 ist ein Beispiel eines JSON Dokuments angeführt.

```
1 {
2   "_id" : ObjectId("1e451b12f5150g12191ea441"),
3   "customer" : "Neuboeck␣Thomas",
4   "date" : "2011/06/01",
5   "article" : "Any␣Milk",
6   "amount": 10
7 }
```

Listing 4.7: JSON (JavaScript Object Notation) Beispiel

Collection

MongoDB Collections sind benannte Gruppierungen von BSON Dokumenten, die man sich als ein äquivalentes Konstrukt zu einer relationalen Tabelle vorstellen kann. Der große Unterschied

zu einer Tabelle einer relationalen Datenbank, liegt im Fehlen einer fixen Schemadefinition für die Collection. Üblicherweise haben die BSON Dokumente innerhalb einer Collection eine idente Struktur, was jedoch nicht zwingend (aufgrund der fehlenden fixen Schemata) notwendig ist. D.h. es ist auch möglich, dass eine Collection aus einer Sammlung heterogener Dokumente besteht. Ein Collection in der MongoDB muss durch den Anwender oder Entwickler nicht manuell erstellt werden. Eine MongoDB Collection wird beim Hinzufügen des ersten Dokuments automatisch miterstellt. Auf Collections kann durch Nutzung der JavaScript-basierten Abfragesprache, wie in Listing 4.8 abgebildet, zugegriffen werden. Die erste Zeile des Beispiels zeigt die MongoDB Abfrage, die restlichen das Ergebnis in JSON Darstellung. Der Zugriff über die vorhandene Java API wird im Zuge der Beschreibung der praktischen Implementierung erklärt.

```
1 > db.sales.find();
2
3 {    "_id" : ObjectId("1e451b12f5150g12191ea441")
4   "customer" : "Neuboeck␣Thomas",
5   "date" : "2011/06/01",
6   "article" : "Any␣Milk",
7   "amount": 10 }
8 {   "_id" : ObjectId("1e451b12f5150g12191ea538"),
9  "customer" : "Neuboeck␣Thomas",
10  "date" : "2011/06/02",
11  "article" : "Any␣Bread",
12  "amount": 3}
```

Listing 4.8: MongoDB Collection - Zugriff via find()

Vor- und Nachteile der Benutzung von MongoDB

Der Einsatz einer dokumentbasierten Datenbank wie der MongoDB, bietet in einigen bestimmten Anwendungsszenarien Vorteile gegenüber dem Einsatz einer relationalen Datenbank. Im Besonderen schaffen dokumentbasierte Datenbanken durch die geringeren verursachten Overhead-Kosten eine ideale Datenbasis zur Speicherung von Ereignisprotokollen. Im Gegensatz dazu bieten relationale Datenbanken Funktionalitäten wie Transaktionen, die Datenspeicherung aufgrund eines Datenbankschemas sowie die Validierung der Daten auf Basis dessen. Solche Funktionalitäten unterstützen vor allem transaktionsorientierte Systeme, welche in anderen Anwendungsszenarien (wie beispielsweise die Protokollierung von Aktivitäten) nicht benötigte Overhead-Kosten verursachen.

Weiters ist in dokumentorientierten Datenbanken die Persistierung der einzelnen Dokumente nicht an ein fix definiertes Schema gebunden, womit auch die Validierung dieses wegfällt. Ein weiterer Vorteil ist eine höhere Flexibilität in der Datenablage, falls dies im jeweiligen Anwendungsszenario wünschenswert ist (andernfalls kann dies natürlich auch ein Nachteil sein). Dies lässt sich am besten an einem einfachen Beispiel erklären: In einer Tabelle werden Ereignisse innerhalb eines Informationssystem mit den Informationen Prozessnummer, Zeitstempel und Aktivitätsbezeichnung protkolliert, welche in weiterer Folge für die Durchführung von Process

Mining Algorithmen Verwendung finden. In einer relationalen Datenbank, wird für solch eine Tabelle ein Schema definiert, mit welchem die einzelnen Informationen als einzelne Spalten der Relation abgebildet werden. Falls sich nun beispielsweise die Anforderung an die Durchführung der Process Mining Techniken verändert, sodass auch der Akteur, welcher an der Ausführung der Aktivitäten beteiligt war, mitprotokolliert wird, bedeutet dies, dass nun entweder das bestehende Tabellenschema angepasst werden oder eine neue Tabelle für diese Informationen angelegt und mit der bestehenden Tabelle verlinkt werden muss. Eine Erweiterung der Tabelle könnte im schlimmsten Fall eine Modifikation der Daten zur Folge haben, was zu zusätzlich notwendigen Aktionen wie einem Datenexport, einer Transformation der vorhandenen Daten sowie zu zusätzlichen Datenimporten führen kann. Solche Zwischenschritte sind nicht nur kostenintensiv sondern auch potentiell fehleranfällig. Dokumentbasierte Datenbanken bieten durch die Eigenschaft der Schemafreiheit mehr Agilität und können flexible Systeme besser unterstützen. Im Falle der Nutzung einer dokumentorientierten Datenbank reicht es aus, das erweiterte Dokument in der bestehenden Collection entsprechend zu ändern, ohne sich über die Struktur der anderen bestehenden Dokumente unnötige Gedanken machen zu müssen. Die erweiterte Struktur der neuen Dokumente hat keine Konsequenz auf den vorhandenen Datenbestand, dieser kann, muss jedoch nicht, angepasst werden.

Ein weiterer Vorteil bei der Verwendung von dokumentorientierten Datenbanken ist, dass zwischen den Dokumenten einzelner Collections keine Beziehungen definiert werden müssen, da diese immer als ein atomares Dokument in einer Collection abgelegt werden. Dies führt natürlich dazu, dass die Prüfung der referenziellen Integrität, welche in relationalen Datenbanken sehr kostenintensiv ist, wegfällt. Ein weiterer positiver Effekt hinsichtlich der Performance bietet die Eigenschaft der Transaktionslosigkeit der MongoDB. Dies erspart zwar unnötige Overhead-Kosten, ist aber störend sollte die entwickelte Applikation gewisse Operationen transaktional abwickeln müssen. Hier bietet die MongoDB keine ausreichende Unterstützung und der Entwickler muss, wenn notwendig, die Transaktionslogik in der zu entwickelten Applikation selbst vorsehen. Für die Ablage von Ereignisprotokollen spielt dieses Kriterium jedoch keine Rolle und wird daher nicht weiter im Detail behandelt.

Die beschriebenen Vorteile des Einsatzes der MongoDB führen dazu, dass die Speicherung von großen Datenmengen sowie die Schreib- und Lesezugriffe auf diese Daten mit einer ungleich besseren Performance als in relationalen Datenbanken durchgeführt werden können. Zusätzlich bietet die MongoDB einen hohen Grad an Skalierbarkeit und konnte mit den neuesten Versionen die Robustheit des Datenbanksystems deutlich steigern.

Natürlich birgt das Fehlen eines definierten Schemas auch potentielle Nachteile. Die MongoDB kann nicht automatisiert entscheiden ob die neu einzufügenden Daten eines Dokuments vollständig bzw. valide sind, d.h. alle notwendigen Felder enthalten sind. Folglich ist es nicht möglich dem Benutzer bzw. Entwickler eine entsprechende Rückmeldung geben oder Fehlermeldung anzuzeigen. Dieser Umstand erfordert bei der Entwicklung von Anwendungen zusätzliche Genauigkeit. Im Besonderen wirkt sich dieser Effekt bei der Verwendung der zu schreibenden bzw. zu lesenden Eigenschaften aus (d.h. es muss, wenn notwendig, innerhalb der Applikation validiert werden). Dieses Problem kann jedoch durch Verwendung von entsprechenden

Schnittstellen in objektorientierten Programmiersprachen, welche die Datenobjekte in die MongoDB persistieren, elegant gelöst werden. Das Morphia Framework bietet solch eine Schnittstelle für objektorientierte Sprachen zur Speicherung von Objekten in die Mongo DB. Das Morphia Framework wird in den folgenden Absätzen noch im Detail vorgestellt.

Eigenschaft	**Vorteile**	**Nachteile**
Datenschema	kein Overhead für Validierung	Prüfung der Vollständigkeit
	Flexibel und Erweiterbar	-
Transaktionen	kein Overhead für Logik	wenn benötigt in Anwendung
Referenzielle Integrität	kein Overhead für Prüfung	-

Tabelle 4.1: Mongo DB: Vor- und Nachteile

Aufgrund der zu erwartenden Performance-Vorteile, im Besonderen im Hinblick auf die potentiell große Datenmenge innerhalb eines Ereignisprotokolls, welche die Ablage des Protokoll in einer dokumentorientierten Datenbank zu einer relationalen Datenbank bietet, kann die Entscheidung zu Gunsten der Verwendung der MongoDB begründet werden. In Tabelle 4.1 sind die Vor- und Nachteile der dokumentbasierten Datenbank gegenüber der Ablage in einer relationalen Datebank zusammengefasst. Die beschriebenen Nachteile der Verwendung der dokumentbasieten Datenbank sind, wie bereits erwähnt, nicht für das geplante Anwendungsszenrio relevant oder können durch die Nutzung von entsprechenden Techniken (z.B.: Morphia Framework als ODM) elegant gelöst werden.

4.3.4 Morphia: Object-Document Mapper

Morphia [48] ist ein Open Source Projekt, welches Funktionalität zum Speichern, Lesen, Löschen und Abfragen von Plain Old JavaObjects (POJOs) in und aus der MongoDB zur Verfügung stellt. Morphia kann als Objekt-Document Mapper (ODM) für die dokumentorientierte Datenbank MongoDB, ähnlich wie JPA (=Java Persistence API) [8] als ORM (=Object-Relational Mapper) fungiert, bezeichnet werden. Mittels Morphia ist es möglich Objekte (z.B.: Java Objekte) in die MongoDB als Dokument zu persistieren sowie vorhandene Dokumente aus Collections abzufragen und zu lesen. Das Ziel, welches bei der Entwicklung von Morphia verfolgt wurde, ist die Stärken von Java und die Funktionalität der MongoDB in einer einfachen, effizienten und offensichtlichen Art und Weise zu verschmelzen. Mittels dem Morphia Framework wird die im vorhergehenden Kapitel beschriebene Problematik der Sicherstellung von Vollständigkeit und Korrektheit der Dokumente explizit durch den Entwickler gelöst. Dies wird ähnlich wie bei JPA oder Hibernate [46] durch die Bereitstellung eines Object-Document Mappers (ODM) ermöglicht. Analog zu JPA oder Hibernate erfolgt dies durch Verwendung von POJOs, welche durch den Mapper in der MongoDB übersetzt und persistiert werden (siehe Abbildung 4.7). Das Morphia Framework ermöglicht es weiters, Abfragen in einer konsistenten Art abzusetzen. Morphia stellt eine Abfrageschnittstelle zur Verfügung, welche die grundsätzliche Mächtigkeit der MongoDB-Abfragen nicht einschränkt, jedoch den Zugriff, generell geltender Java Prinzipien bzw. gemäß praktischen Erfahrungen, in einfacherer und verständlicherer Form ermöglicht.

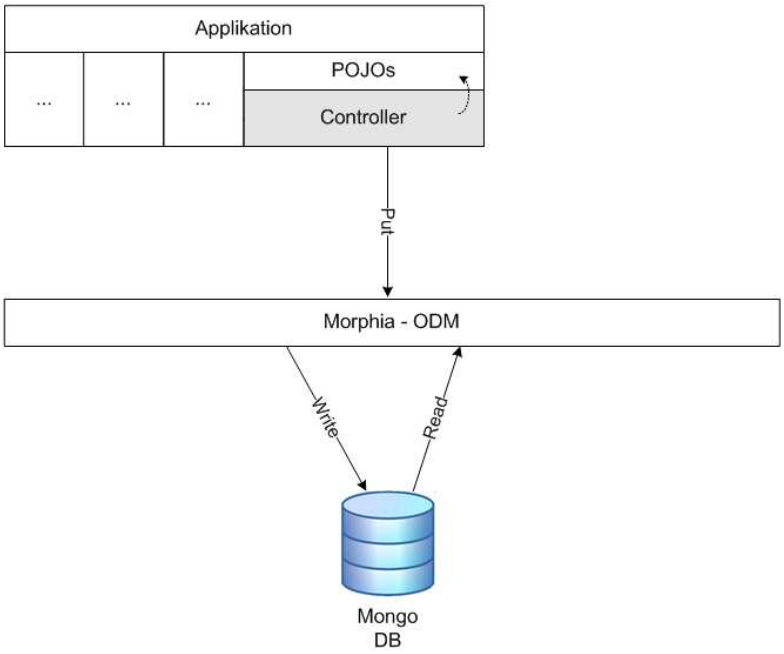

Abbildung 4.7: Morphia - Schematische Darstellung der Funktionsweise des Objekt-Dokument Mappers

Die Ziele des Morphia Frameworks lassen sich wie folgt zusammenfassen:

- Typsicherheit: Durch die Verwendung von POJOs
- Konfiguration via Annotationen: Keine XML Konfiguration oder zusätzlicher JAVA-Code
- Einfache, intuitive Abfrageschnittstelle
- Flexibles Domänenmodell
- Kompatibel mit Dependency Injection Frameworks

Die Entwicklung von POJOs mit Morphia, zur Speicherung in der dokumentbasierten Datenbank MongoDB, gestaltet sich ähnlich der Entwicklung mit JPA oder Hibernate. Im ersten Schritt muss das gewünschte Java Objekt mit entsprechenden Annotationen versehen werden. Das im Listing 4.9 aufbereitete Beispiel zeigt, wie ein POJO für eine Klasse *Employee* annotiert werden kann. In diesem einfachen Fall wird das Objekt selbst als *@Entity(„employee“)*

annotiert, was bedeutet, dass das Objekt als Dokument in der Collection *employee* in der MongoDB abgelegt wird. Zusätzlich ist im Beispiel das Feld *id* vom Typ *ObjectID* als *@Id* annotiert. *ObjectID* ist ein spezieller BSON Datentyp, welcher als Binärwert mit einer Länge von 12 Byte repräsentiert wird. Die *ObjectID* setzt sich aus dem Zeitstempel des Zeitpunkts der Erzeugung (4 Byte), der Maschinen ID (3 Byte), einer Prozess ID (2 Byte) und einem internen Zähler (3 Byte) zusammen. Alle von der MongoDB offiziell unterstützten Treiber verwenden den Typ *ObjectID* als Standard-Variante für die Eigenschaft *id*. Verwendet man anstatt des Typs *ObjectID*, eine andere unterstützte Variante (z.B.: UUID, int, ...) muss der entsprechende ID-Wert direkt vom Entwickler gesetzt werden. Alle restlichen nicht-annotierten Felder des Objekts werden automatisch von Morphia in die MongoDB entsprechend zugeordnet. Will man bestimmte Felder nicht mappen, können diese mit der Annotation *@Transient* explizit ausgeschlossen werden. Morphia stellt noch weitere Annotationsmöglichkeiten zur Verfügung (bei der Entwicklung hatte man sich stark an JPA orientiert). Näheres kann auf der Morphia Projektseite [48] nachgelesen werden.

```
1  import com.google.code.morphia.annotations.Entity;
2  import com.google.code.morphia.annotations.Id;
3  import org.bson.types.ObjectId;
4
5  @Entity("employees")
6  public class Employee {
7      @Id private ObjectId id;
8
9      private String forename;
10     private String surename;
11     private Date   birthdate;
12     private double salary;
13 }
```

Listing 4.9: Annotiertes Morphia POJO

Um ein annotiertes Objekt in der MongoDB abzulegen, ist es notwendig, ein entsprechendes *Morphia*-Objekt zum Zugriff auf die Datenbank zu erstellen. In Listing 4.10 wird die notwendige Systematik an einem Beispiel skizziert. Es ist notwendig, sowohl eine *Morphia*- als auch eine *Mongo*-Objektinstanz zu erstellen. Der *Mongo*-Objektinstanz müssen sowohl der Hostname als auch der Port der zu verwendenden MongoDB-Datenbankinstanz übergeben werden. Mittels dem *Morphia*-Objekt (fungiert als Factory) wird, durch Übergabe der *Mongo*-Objektinstanz sowie des gewünschten Datenbanknamens, eine *DataStore*-Objektinstanz erstellt. Dieses *DataStore*-Objekt wird in Folge zum direkten Zugriff auf die MongoDB-Datenbankinstanz verwendet. Das *Datastore*-Objekt stellt eine typsichere Zugriffsvariante auf die Dokumente in der MongoDB dar. Der *Datastore* stellt Operationen zum Lesen (*get*), Finden (*find*), Schreiben (*save*), Aktualisieren (*update*) und Löschen (*delete*) der als Dokumente gemappte Java-Objekte in der MongoDB zur Verfügung. Durch den Aufruf der *save()* Methode des *Datastore-Objekts* mit dem jeweilig zu mappenden Objekt als Parameter, kann das zu mappende Objekt als Dokument in der MongoDB persistiert werden.

```
1 import com.google.code.morphia.Datastore;
2 import com.google.code.morphia.DatastoreImpl;
3 import com.google.code.morphia.Morphia;
4 import com.mongodb.Mongo;
5
6 Morphia morphia = new Morphia();
7 Mongo mongo = new Mongo( hostname, port );
8 Datastore ds = morphia.createDatastore( mongo , databaseName );
9
10 Employee employee =
11     new Employee (forename, surename, birthdate, salary);
12 ds.save(employee);
```

Listing 4.10: Morphia Datenbankzugriff

Der Zugriff auf bestehende Dokumente in einer MongoDB-Collection, kann über die bereits erwähnte Abfrageschnittstelle von Morphia erfolgen. Abfragen können direkt mit der *get()* Methode, wenn beispielsweise der eindeutige Bezeichner bekannt ist, oder über die *find()* Methode erfolgen (siehe Listing 4.11). Die *find()* Methode kann verwendet werden, um alle Dokumente einer Collection aufzufinden oder wie im Beispiel illustriert, durch die Angabe einer Restriktion nur den betroffenen Teil der Dokumente abzufragen.

```
1 Datastore ds = morphia.createDatastore( mongo , databaseName );
2 Employee employee = ds.get(Employee.class, employeeId);
3
4 List<Employee> poorEmployees =
5       ds.find(Employee.class,    "salary < ", 2000).asList();
```

Listing 4.11: Morphia: Collection-Zugriff mittels find()

In der praktischen Umsetzung von Anwendungen mit Zugriff auf eine Datenschicht, ist es eine gängige Strategie den Code, welcher den Zugriff auf die Datenschicht abbildet, in ein eigenes Data Access Object (DAO) [23] zu kapseln. Dies bedeutet, dass die ausführende Klasse (beispielsweise ein Controller-Objekt) eine Instanz des DAO enthält. DAOs ermöglichen Klassen, welche kein Wissen über die Datenschicht sowie der darin enthaltenen Strukturen enthalten, den Zugriff auf die Datenschicht. Dies bedeutet, dass der Entwickler sich bei der Implementierung, solch einer Controller-Klasse, keine expliziten Gedanken über den konkreten Datenbankzugriff machen muss. Morphia bietet für die Entwicklung von DAOs per Standard eine abstrakte Implementierung eines *BasicDAOs*, welches die abstrakten Methoden für den Datenbankzugriff generisch abbildet und als Basisklasse für eine eigene Implementierung genutzt werden kann. Listing 4.12 zeigt eine einfache DAO Implementierung für das in diesem Kapitel beschriebene Employee-Beispiel. Zu beachten ist, dass solch ein DAO um beliebige individuelle Zugriffsmethoden oder Abfragen (z.B.: um den dienstältesten Mitarbeiter zu erhalten) erweitert werden kann. Die Abfragelogik ist entsprechend im DAO abzubilden (unter Nutzung der abstrakten Standard-Implementierung) und kann von außen abgefragt werden, ohne dass die Konsumenten die Abfragesyntax oder Datenbankstruktur kennen müssen.

```
1 import com.google.code.morphia.Morphia;
2 import com.google.code.morphia.dao.DAO;
3 import com.mongodb.DBCollection;
4 import com.mongodb.Mongo;
5
6 public class EmployeeDAO extends BasicDAO<Employee, String> {
7     public EmployeeDAO(Morphia morphia, Mongo mongo ) {
8         super(mongo, morphia, "employeeDB" );
9     }
10 }
11
12 EmployeeDAO employeeDAO = new EmployeeDAO(morphia,mongo);
13 employeeDAO.save(
14       new Employee (forename, surename, birthdate, salary));
```

Listing 4.12: Morphia: Einfache DAO-Implementierung

4.3.5 Protokollierungsmechanismus: Morphia Camel Tracer

Der *Morphia Camel Tracer* stellt eine Erweiterung des bereits beschriebenen *Apache Camel Tracer*-Mechanismus dar. Mit der vorhandenen *Apache Camel Tracer*-Implementierung, ist die Protokollierung der auftretenden Ereignisse nur auf der Standardausgabeschnittstelle oder in eine relationale Datenbank (die bereits erwähnte JPA-Implementierung) möglich. Aufgrund der Entscheidung, die dokumentorientierte MongoDB als Datenbasis für das Ereignisprotokoll zu verwenden, ergab sich die Notwendigkeit einen entsprechend erweiterten *Apache Camel Tracer*-Mechanismus zu implementieren. Für den Datenbankzugriff wurde der Object-Document Mapper Morphia (wie oben beschrieben) verwendet, um die Lösung ähnlich der JPA-Implementierung aufzubauen (diese verwendet ORM) und die bereits beschriebenen Vorteile solch eines Mapping-Mechanismus nutzen zu können. Die Lösung wurde analog zur bestehenden *Apache Camel*-Implementierung vollkommen in der Programmiersprache Java implementiert.

Der entwickelte Protokollierungsmechanismus *Morphia Camel Tracer* wurde, wie im Klassendiagramm 4.8 abgebildet, umgesetzt. Im Zuge der Entwicklung, wurden die von *Apache Camel* verfügbaren Schnittstellen verwendet und somit der vorhandene *Camel Tracer*-Mechanismus um eine Morphia-Implementierung des *Tracers* erweitert. Die Schlüsselelemente des entwickelten Protokollierungsmechanismus *Morphia Camel Tracer* lassen sich wie folgt gliedern:

- **MorphiaTraceEventMessage:** Morphia POJO (Plain old Java object) - Repräsentiert ein Prozessereignis und wird via Morphia in der MongoDB persistiert
- **MorphiaTraceEventMessageDAO:** Morphia Data Access Object (DAO) - Zugriffsschnittstelle auf das Ereignisprotokoll via Morphia
- **MorphiaTraceEventHandler:** Implementierung eines Apache Trace Handlers für die Morphia Implementierung

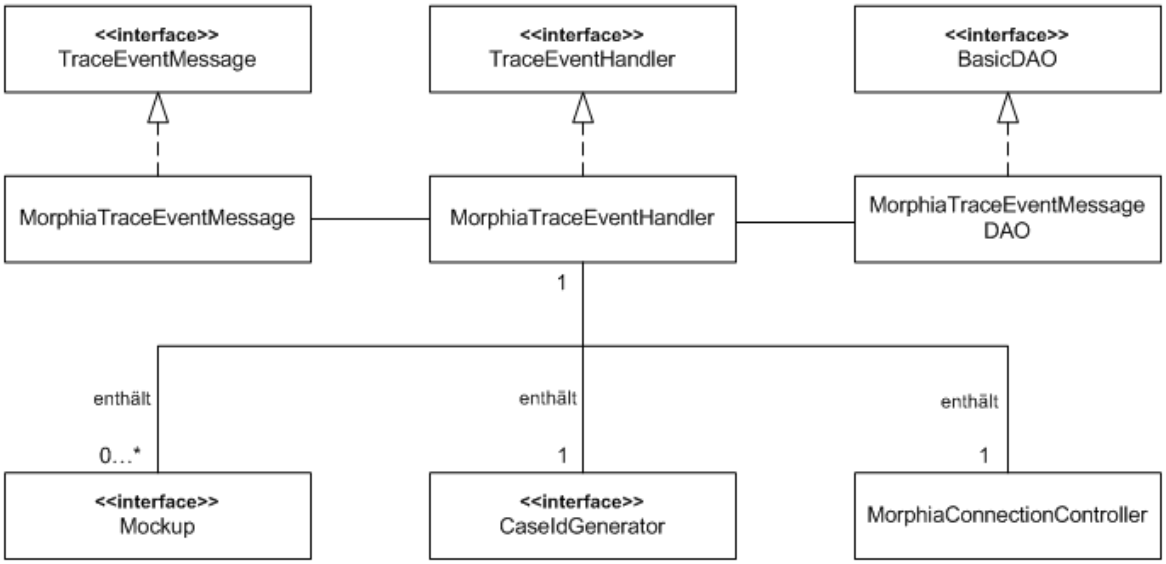

Abbildung 4.8: Klassendiagramm Morphia Camel Tracer

- **MorphiaConnectionController:** Klasse zur Verwaltung der Morphia-Verbindung zur MongoDB
- **CaseIdGenerator:** Schnittstelle um IDs für Prozessinstanzen zu generieren
- **Mockup:** Schnittstelle um Mockup-Logiken in die Morphia Camel Tracer-Implementierung zu integrieren.

Im weiteren Verlauf werden diese Schlüsselemente der Implementierung und die im Laufe der Umsetzung getroffenen Entscheidungen im Detail beschrieben. Abschließend wird die Konfiguration sowie die Vorrausetzungen für die Verwendung des *Morphia Camel Tracers* in einer *Apache Camel* Anwendung besprochen. Im Zuge dessen werden auch nähere Detailinformationen zur veröffentlichten Lösung gegeben und die notwendigen Schitte zur Integration dieser in eine Applikation via Apache Maven [43] erklärt.

MorphiaTraceEventMessage

MorphiaTraceEventMessage ist als Plain Old Java Object (POJO) realisiert. In diesem POJO sind die Daten über ein, vom *Apache Camel Tracer* abgefangenes, Ereignis enthalten und kann folglich zur Persistierung mittels eines ODMs in die Datenbank verwendet werden. Aktivitäten einer *Apache Camel Route* (Prozessinstanz) werden mittels des *Morphia Tracers*, durch Verwendung dieses POJOs, in einem Ereignisprotokoll (MongoDB) abgebildet. Für den Datenbankzugriff auf die Datenbankinstanz der MongoDB sowie zur Persistierung der Protokolldaten, wird Morphia als Objekt-Dokument Mapper verwendet. Daher war es notwendig ein

Eigenschaft	Datentyp	Beschreibung
ID	ObjectID	Eindeutige Message ID generiert durch die Datenbank
Timestamp	Date	Zeitstempel des Ereignisses. Systemzeit der JVM.
FromEndpointUri	String	Die URI des Consumers welcher die *Route* gestartet hat.
PreviousNode	String	ID des vorhergehenden Schrittes der *Route*. Null beim ersten Schritt.
ToNode	String	ID des nächsten Schrittes der *Route*.
InternalCaseID	String	Intere Prozessinstanz-ID des MorphiaTraceEventHandlers.
CorrelationID	String	Correlation-ID des aktuellen Exhanges.
ExchangeId	String	Eindeutige ID des aktuellen Exhanges.
ShortExchangeId	String	Eindeutige ID des aktuellen Exhanges ohne Maschinenamen.
ExchangePattern	String	Verwendetes Exchange Pattern - beispielsweise InOut oder InOnly.
Properties	String	Eigenschaften des Exchange.
Headers	String	Header-Informationen der In-Message.
Body	String	Body der In-Message.
BodyType	String	Typinformation zum Body der In-Message.
OutHeaders	String	Header-Informationen der Out-Message.
OutBody	String	Body der Out-Message.
OutBodyType	String	Typinformation zum Body der Out-Message.
CausedByException	String	Exception des Exchanges (wenn vorhanden) inklusive StackTrace.

Tabelle 4.2: MorphiaTraceEventMessage: Eigenschaften des POJOs [9]

eigenes Morphia POJO, analog zur JPA Implementierung des *Tracers*, zu implementieren, welches die Prozessdaten eines Ereignis enthält. Die entwickelte Klasse implementiert das Interface *org.apache.camel.processor.interceptor.TraceEventMessage*, um vom bestehenden *Apache Camel Tracer*-Mechanismus entsprechend verwendet werden zu können. Ein *MorphiaTraceEventMessage* Objekt enhält Informationen zum Ausführungszeitpunkts der protokollierten Aktion, zum betroffenen Ausführungsschritt der *Route* (z.B.: *Camel Processor*), eine *ExchangeID* sowie eine eindeutige *MessageID*.

In der Tabelle 4.2 sind die einzelnen Eigenschaften des POJOs aufbereitet dargestellt. Zuzüglich zur Standardimplementierung enthält das Objekt auch noch die Information zu einer internen Prozessinstanz-ID (InternalCaseID), welche im Zuge der Beschreibung des Ablaufes eines Protokollierungsdurchlaufes des *Tracers*, noch im Detail erklärt wird. Die bereitgestellte Klasse *MorphiaTraceEventMessage*, welche mit den notwendigen Morphia Annotationen beschrieben ist, kann mittels dem Morphia-Mechanismus zum Persistieren oder Laden von Objekten aus der MongoDB verwendet werden.

MorphiaTraceEventMessageDAO

Wie bereits erwähnt ist es eine Best-Practice Lösung, den Zugriff auf die Datenschicht eines Systems in ein zentrales Data Access Object (DAO) zu kapseln. Da diese Strategie durch Morphia explizit unterstützt wird, indem eine abstrakte Basisimplementierung eines DAOs be-

reitgestellt wird, wurde im Zuge der praktischen Umsetzung ein *MorphiaTraceEventMessage-DAO* implementiert. *MorphiaTraceEventMessageDAO* ist eine Ableitung der abstrakten Klasse *com.google.code.morphia.dao.BasicDAO*, welche von Morphia zur Verfügung gestellt wird. Das *BasicDAO* macht sich das Prinzip der Generezität zu Nutze, um jegliche Subklassen, unabhängig vom konkreten Implementierungstyp des zu verwendenden POJOs, zu erlauben. Das *Morphia-TraceEventMessageDAO* stellt Methoden zum Zugriff auf die Datenbank, für die bereits beschriebene *MorphiaTraceEventMessage* zur Verfügung. Die verfügbaren Methoden lassen sich in Lese- und Schreibzugriffe wie in Tabelle 4.3 abgebildet unterteilen.

Methode	**Typ**	**Beschreibung**
saveMessage(Object entity)	Write	Speichert eine Message in MongoDB.
findAll( )	Read	Liste aller Messages der Datenbank.
findAll(Date min, Date max)	Read	Liste von Messages eines Zeitraumes.
findMessageWithMinTimestamp()	Read	Retouniert die älteste Message.
findMessageWithMaxTimestamp()	Read	Retouniert die jüngste Message.
getFirstInternalCaseID()	Read	Retouniert die kleinste interne ID.
getLastInternalCaseID()	Read	Retouniert die größte intere ID.
internalCaseIDExists(String internalCaseID)	Read	Prüft ob interne ID in der Datenbank existiert.

Tabelle 4.3: MorphiaTraceEventMessageDAO: Öffentliche Methoden

MorphiaConnectionController

Um die Verbindungen zur Ereignisprotokoll-Datenbasis sowie die dafür notwendigen Verbindungsinformationen zu kapseln, wurde ein *MorphiaConnectionController* entwickelt. Aufgabe dieser zentralen Klasse ist es die notwendigen Objektinstanzen zum Datenbankzugriff zu erstellen und zu verwalten (Mongo, Morphia sowie Instanzen des DAOs) sowie allgemein notwendige Datenbankabfragen (Bsp.: Datenbankliste) zur Verfügung zu stellen.

Die notwendigen Verbindungsinformationen werden in getrennten Klassen verwaltet: a) *MongoDBConnectionInfo* und b) *MongoDBCredentials*. Die resultierende Klassenstruktur ist in Abbildung 4.9 dargestellt. *MongoDBConnectionInfo* enthält die zentralen Verbindungsparameter wie den Hostname, den Port sowie den Datenbanknamen der MongoDB-Datenbankinstanz in welche das Ereignisprotokoll abgelegt werden soll. *MongoDBCredentials* enthält die Authentifikationsparameter für die definierte MongoDB Datenbankinstanz. Dies ist nur dann relevant, wenn die Datenbankinstanz in einem nicht anonymen Modus läuft, d.h. wenn ein Datenbankzugriff nur authentifiziert erfolgen darf bzw. soll. Der *MorphiaConnectionController* verwendet diese Informationen, um die Verbindung zur Datenbank herzustellen und in Folge eine Objektinstanz der *MorphiaTraceEventMessageDAO*-Klasse für den weiteren Datenbankzugriff zu erzeugen.

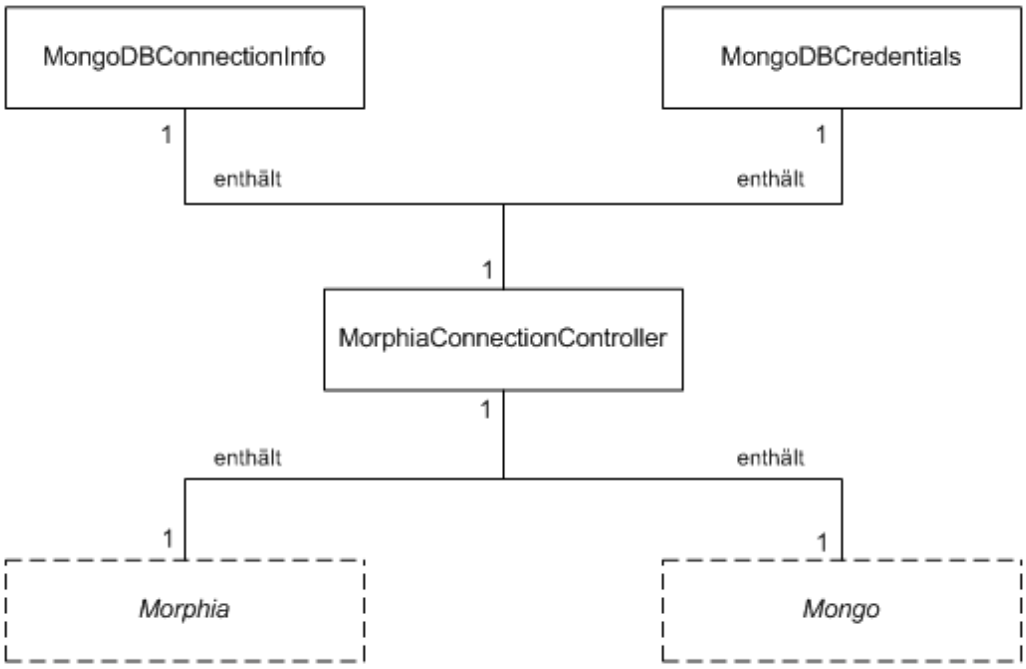

Abbildung 4.9: Klassendiagramm Morphia Camel Tracer

CaseIdGenerator

Mittels des *CaseIDGenerators* wird die Möglichkeit zur Erstellung von IDs für die weitere Verwendung im *MorphiaTraceEventHandler* zur Verfügung gestellt. Die Problematik, woraus die Anforderung zur Implementierung solch eines Generators enstanden war, wird im Zuge des *MorphiaTraceEventHandler* im Detail erklärt. *CaseIdGenerator* (Abbildung 4.10) ist eine Schnittstelle, welche lediglich eine Methode zur Erstellung einer neuen ID zur Verfügung stellt. Diese Methode erlaubt es dem Generator die zuletzt generierte ID als Parameter zu übergeben (z.B.: ID als Sequenz) und retourniert eine neu generierte ID an das aufrufende Objekt. Die Sicherstellung der Eindeutigkeit der generierten IDs ist nicht Aufgabe des Generators, da der implementierte Generator über keine Kenntniss (z.B.: in Form einer Liste) der bereits erzeugten IDs verfügt. Theoretisch wäre der Generator um eine Liste von bereits erzeugten IDs erweiterbar, jedoch müsste hierfür der Generator beim Start der jeweiligen Applikation einen Abgleich mit der Datenbasis durchführen. Um den Generator so generisch wie möglich zu halten sowie um eine direkte Beziehung und folglich eine zu enge Kopplung zu vermeiden, wurde von der Implementierung einer solchen Erweiterung abgesehen. Daher wurde die Verantwortung zur Sicherstellung der Eindeutigkeit der generierten IDs an die aufrufende Objektinstanz (in unserem Fall der *MorphiaTraceEventHandler*) übergeben. Die aktuelle Lösung stellt zwei Implementie-

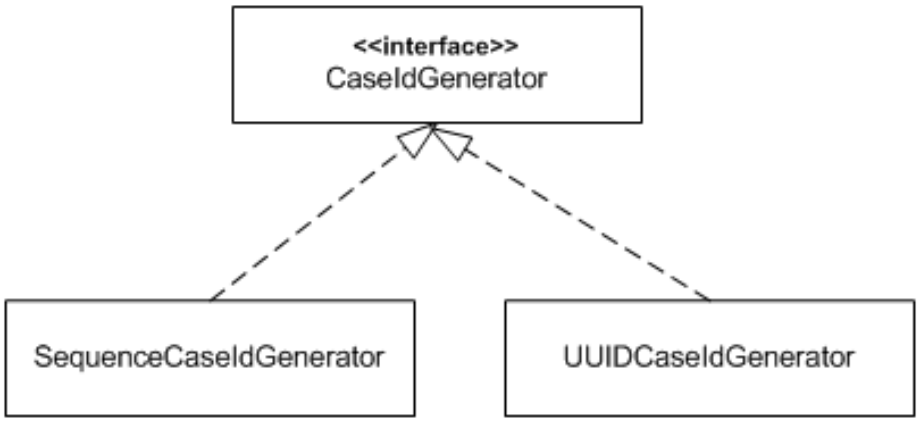

Abbildung 4.10: Klassendiagramm CaseIdGenerator

rungen der *CaseIdGenerator*-Schnittstelle zur Verfügung. Der *SequenceCaseIdGenerator* generiert aufgrund der letzt bekannten ID die nächst mögliche ID (z.B.: zuletzt generierte ID: 1001 => Generator erzeugt ID 1002). Hierfür ist es notwendig, dass der Aufrufende dem Generator die zuletzt generierte ID zur Verfügung stellt. Der *UUIDCaseIDGenerator* erstellt eine neue Universally Unique Identifier (UUID) ohne die zuletzt erstellten ID zu berücksichtigen. Es ist ausschließlich die Nutzung eines *CaseIdGenerators* (parallele Verwendung ist nicht erlaubt) im *MorphiaTraceEventHandler* möglich, die hierzu notwendige Konfiguration wird zum Abschluss des Teilkapitels Protokollierungsmechanismus erklärt.

Mockup

Zusätzlich wurde Funktionalität implementiert, mit welcher zusätzliche Aktionen im Zuge der *Apache Camel Tracer*-Logik, d.h. während Behandlung von *Exchange* und *Messages* einer *Route* durch den *TraceHandler*, zu simulieren. Dies soll die Möglichkeit schaffen, spezielles Verhalten oder Prozessinformationen, welche aktuell im System nicht vorhanden sind, in den protokollierten Ereignissen aufgrund einer Simulation abzubilden. Ein Beispiel hierfür wäre ein System, welches aktuell noch keine Informationen über den für die Aktivität verantwortlichen Akteur in der *Message* mitführt, im Zuge einer Simulation um genau solche Informationen zu erweitern. Dies ermöglicht es auch schon vor einer eventuell geplanten Anpassung des Systems, welche die Protokolldaten erzeugt, entsprechend Daten oder Verhalten mit Hilfe eines Mockups zu erzeugen. Eine auf die Ereignisprotokolldaten basierende Implementierung (z.B.: einen Export/Importmechanismus oder eine Process Mining Technik) könnte somit um zusätzliche Informationen erweitert bzw. angepasst und auf diese Änderungen getestet werden. Im Zuge der praktischen Umsetzung hatten wir mit einem Quellsystem gearbeitet, welches in der

aktuellen Version keine Akteur-Informationen zur Verfügung stellte. Da wir den Importmechanismus (dies wird im weiteren Verlauf besprochen) jedoch auch auf die Berücksichtigung von Informationen zu den Akteuren (z.B.: um im weiteren die Organisatorische Perspektive des Prozessmodells betrachten zu können) ausgelegt hatten, fiel der Entschluss auf die Umsetzung solch eines Mockups zur Erzeugung von Simulationsdaten.

Der *MorphiaTraceEventHandler* führt eine Liste von möglichen Mockups, welche gleich nach Erhalt eines neuen Ereignisse auf den *Exchange* bzw. die *Message* angewandt werden. Alle Mockups müssen das Interface *Mockup* (siehe 4.13) implementieren, welches eine Methode *execute* zur Verfügung stellt. Die Mockup Implementierung *ResourceMockup* (Abbildung 4.11) ergänzt den *Message*-Header, des jeweiligen *Message*-Objekts eines Ereignisses, um eine Akteur-Information (hierzu wird eine Information aus einer statischen Liste zufällig ausgewählt). Die statische Liste an möglichen unterschiedlichen Akteur-Informationen, das für die Information zu verwendende Header-Field (der *Message*) sowie die Wahrscheinlichkeit, dass einer Akteur-Information verwendet werden soll (die Logik erlaubt es, dass auch gar keine Information gesetzt wird), sind fix im Mockup als konstante Informationen hinterlegt. Die Bezeichnung *ResourceMockup* wurde gewählt, da in der Zielrepräsentation der Protokolldaten die Akteur-Information als Resource-Information bezeichnet wird. Daher sind in weiterer Folge die Bezeichnungen Akteur und Resource als ident zu betrachten. Sollten andere Testdaten oder Simulationsverhalten notwendig sein, kann ein neues oder ergänzendes Mockup implementiert werden. Die Konfiguration von Mockups zur Nutzung in einer Applikation wird im folgenden Kapitel erklärt.

```
1 package org.apache.camel.processor.interceptor.morphia.mockups;
2
3 import org.apache.camel.Exchange;
4
5 public interface Mockup {
6     public void execute(Exchange exchange);
7 }
```

Listing 4.13: MorphiaCamelTracer - Mockup Interface

MorphiaTraceEventHandler

Wie im Kapitel 4.3.2 besprochen, erlaubt *Apache Camel* mittels des *Tracer*-Mechanismus die Protokollierung der eingetretenen Ereignisse durchgeführter *Routes*. Ein *TraceEventHandlers* erlaubt es spezifische Logik (z.B.: eigener Protokollierungsmechanismus) für die Behandlung dieser Ereignisse bereitzustellen. Um den bestehenden *Apache Camel Tracer-Mechanismus* für die Protokollierung der Ereignisse in die MongoDB nutzen zu können, war es notwendig (wie bereits besprochen) eine entsprechende Erweiterung zu implementieren. Der erste Grundstein wurde mit der Umsetzung eines passenden POJOs zur Darstellung der Ereignisdaten gelegt. Basierend auf dieser Entwicklung sowie der von *Apache Camel* zur Verfügung gestellten Schnittstelle wurde ein passender *MorphiaTraceEventHandler* implementiert, welcher die Behandlung der innerhalb einer *Camel Route* auftretenden Ereignisse übernimmt. Um einen entsprechenden *TraceHandler* zu entwickeln, ist es notwendig, das Interface

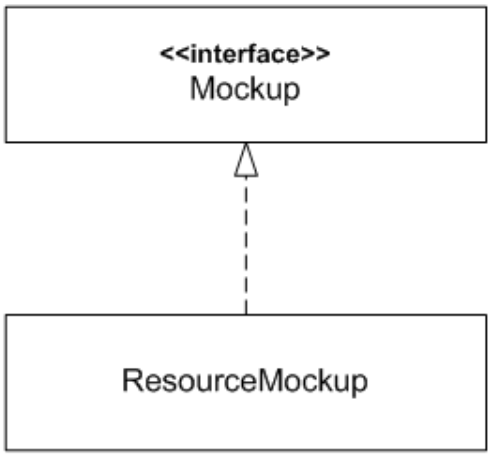

Abbildung 4.11: Klassendiagramm Mockup

org.apache.camel.processor.interceptor.TraceEventHandler zu implementieren und die dadurch notwendigen Methoden umzusetzen (siehe Tabelle 4.4.

Methode	**Beschreibung**
traceExchange	Aufgerufen bevor Exchange verarbeitet wird (bei Pattern InOnly).
traceExchangeIn	Aufgerufen bevor Exchange verarbeitet wird (bei Pattern InOut).
traceExchangeOut	Aufgerufen nachdem Exchange verarbeitet wurde (bei Pattern InOut).

Tabelle 4.4: TraceEventHandler Interface: Methoden

Der entwickelte *TraceHandler* kann dem bestehenden *CamelContext* zugewiesen und folglich beim Eintritt eines Ereignisses aufgerufen werden, mit der Aufgabe entsprechende Aktionen (z.B.: wie in unseren Fall die Protokollierung) durchzuführen. Bei jedem Aufruf des *TraceHandlers*, werden diesem Informationen (in Form von Parametern) zum entsprechenden Ereignis übergeben (siehe 4.4 - Methoden des Interface *TraceEventHandler*). Für unsere Implementierung war die wichtigste Information das *Exchange*-Objekt, welches wie im Teilkapitel *Apache Camel* bereits beschrieben ein *Apache Camel Message*-Objekt (In-*Message*) kapselt. Die aus dem *Exchange*-Objekt sowie dem inkludierten *Message*-Objekt gewonnen Informationen, werden in das bereits besprochene *MorphiaTraceEventMessage*-Objekt überführt (siehe Transformation Tabelle 4.5) und als Dokument in der MongoDB abgelegt.

Der Verarbeitungsablauf des *MorphiaTraceEventHandler* ist in Abbildung 4.12 dargestellt. Der Ablauf lässt sich in mehrere Phasen gliedern:

- Verbindungsaufbau zur Datenbankinstanz
- Durchführung von Mockup-Logiken
- Ermittlung eindeutiger internen ProzessID

Zieleigenschaft	Quellobjekt	Quelleigenschaft
Timestamp	InMessage	Timestamp
PreviousNode	Exchange	PreviousNode
ToNode	ProcessorDefinition	ID
InternalCaseID	InMessage	Wenn erste Message generiert, sonst Header.InterneID
CorrelationID	Exchange	Properties.CorrelationID
ExchangeId	String	Properties.ExchangeID
Properties	InMessage	Properties
Headers	InMessage	Headers
Body	InMessage	Body

Tabelle 4.5: TraceEventHandler: Transformation Prozessdaten in MorphiaTraceEventMessage (nur relevante)

- Transformation der Ereignisdaten in das POJO
- Persistierung des POJOs in die MongoDB.

Der *Verbindungsaufbau zur Datenbankinstanz* 4.12 erfolgt durch Nutzung des besprochenen *MorphiaConnectionControllers*, welcher ein DAO für den Datenbankzugriff erstellt (Listing 4.14).

```
1 MorphiaTraceEventMessageDAO traceMsgDAO =
2         this.morphiaController.createMorphiaTraceEventMessageDAO(true);
```

Listing 4.14: Erstellen von MorphiaTraceEventMessageDAO

Die *Durchführung der Mockup-Logiken* 4.12 löst, wie zuvor erwähnt, alle konfigurierten Mockups in sequentieller Reihenfolge (abhängig von der Reihenfolge in der Konfigurationsdatei) aus. Im nächsten Schritt *Ermittlung eindeutiger internen Prozess-ID* 4.12 wurde ein Mechanismus zur Erzeugung einer internen Prozessinstanz-ID implementiert. Die Analyse- und Testläufe der ersten Implementierung des *Morphia Camel Tracers* zeigten, dass die verfügbare *ExchangeID* innerhalb einer *Route* nicht zwangsläufig für alle aufgetretenen Ereignisse ident sein muss (kann für manche *Routes* jedoch stimmen, dieses Verhalten ist jedoch nicht verlässlich verfügbar). Weitere Analysen zeigten, dass auch keine andere Information verfügbar war, mit welcher wir eine vollständige Verbindung zwischen einer Prozessinstanz und den zugehörigen Ereignissen herstellbar ist. Die Analyse zeigte, dass diese Problematik im Besonderen bei Parallelitäten sowie folglich notwendigen Aggregationen innerhalb der *Camel Route* auftraten. Diese Herausforderung *Unvollständige Verbindungen zwischen Prozessinstanzen und Ereignissen* wurde bereits im Kapitel 2.6 besprochen.

Im umgesetzten *MorphiaTraceEventHandler* konnte das Problem der fehlenden eindeutigen Prozessinstanz-ID durch die Einführung einer internen Prozessinstanz-ID gelöst werden. Für diese Umsetzung wurde die Annahme getroffen, dass innerhalb einer *Route* die Header-Informationen einer *Camel-Message* auch in den folgenden *Camel-Messages* verfügbar sind

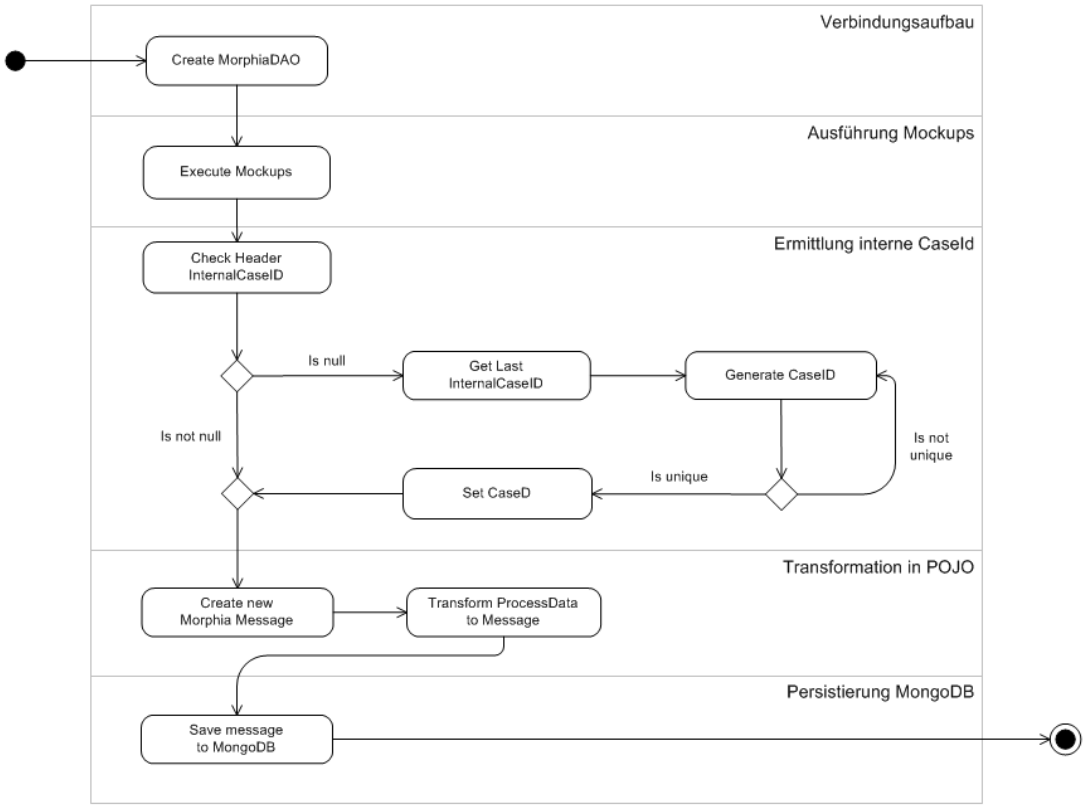

Abbildung 4.12: Ablaufdiagramm MorphiaTraceEventHandler

(außer diese werden bewusst entfernt). D.h. wenn in der ersten zu protokollierende *Message* (d.h. erster *Processor* welcher durchgeführt wird) einer *Route* dem *Message*-Header eine Prozessinstanz-ID hinzugefügt wird, ist diese auch in den folgenden *Messages* verfügbar. Diese Annahme erlaubte es, eine ID-Verwaltungs- und Erzeugungslogik zu implementieren, welche der ersten auftretenden *Message* einer *Route* eine interne ID hinzufügt.

Im Zuge der Umsetzung waren zwei Probleme zu lösen: 1) Wie kann die erste zu protokollierende *Message* einer *Route* (Prozessinstanz) identifiziert werden? und 2) Wie kann die Eindeutigkeit der Prozessinstanz-ID sichergestellt werden? Zur Identifikation der ersten *Message* einer *Route*, war die bereits getroffene Annahme, dass die Header-Information einer *Message* auch in allen folgenden *Messages* der *Route* verfügbar ist, hilfreich. Dies bedeutet, dass es ausreichend ist zu prüfen, ob im Header der *Message* eine interne Prozessinstanz-ID verfügbar ist. Ist dies nicht der Fall, dann wird eine neue Prozessinstanz-ID erzeugt. Ansonsten ist die zu behandelnde *Message* nicht die erste und die im Header vorhandene ID wird beibehalten. Die Thematik der Eindeutigkeit der Prozessinstanz-ID ist nur durch Abgleich einer neu generierten ID mit dem bestehenden Ereignisprotokoll lösbar (dies betrifft nur die Implementierung des *Se-*

quenceCaseIdGenerator und nicht des *UUIDCaseIdGenerator*). Dies bedeutet, dass mittels des verfügbaren *CaseIDGenerator* eine neue ID erzeugt wird und diese via des *MorphiaTraceEvent-MessageDAO* auf Eindeutigkeit geprüft wird (Listing 4.15). Sollte diese nicht eindeutig sein, muss eine neue ID erzeugt werden und die Prüfung auf Eindeutigkeit wiederholt durchgeführt werden. Ist sie aber eindeutig, wird diese dem Header der *Message* zugewiesen.

```
1 traceMsgDAO.internalCaseIDExists(generatedInternalCaseID.toString();
```

Listing 4.15: Prüfen ob interne CaseID schon verwendet wurde

Im Schritt *Transformation der Ereignisdaten in das POJO* 4.12 werden die vorhandenen Prozessinformation in die *MorphiaTraceEventMessage* überführt (siehe Transformationstabelle 4.5) und im abschließenden Schritt *Persistierung des POJO in die MongoDB* 4.12 dauerhaft in der entsprechenden MongoDB Collection abgespeichert (Listing 4.16).

```
1 traceMsgDAO.saveMessage(morphiaMsg);
```

Listing 4.16: Persistieren der MorphiaTraceEventMessage

Konfiguration für Apache Camel Projekt

Die entwickelte Lösung *Morphia Camel Tracer* kann via des *Camel* XML-Konfigurationsfile (camel-context.xml) in ein bestehendes *Apache Camel*-Projekt integriert werden. Die verwendete Konfigurationvariante (Konfiguration via *Spring XML*) macht sich das Konzept der Dependency Injection zu nutze, um die konfigurierten Objekte in den *CamelContext* zu laden. Folgend wird die Konfiguration anhand des Beispiels 4.17 erklärt. Die Grundlagen zur *Spring* Konfiguration [52] sowie zum Konzept der Dependency Injection, werden in dieser Arbeit nicht vertiefend erklärt und sind in den vorhandenen Dokumentationen, welche auf den Projektseiten verfügbar sind, nachzulesen. Das abgebildete Beispiel zeigt alle konfigurierbaren Komponenten des *MorphiaCamelTracer*. Es ist notwendig eine Bean der Klasse *MorphiaTraceHandler* zu definieren. Diese Bean muss nun der entsprechenden Eigenschaft *TraceHandler* der zugehörigen *Tracer*-Bean zugewiesen werden. Für die definierte Bean müssen die Verbindungsinformationen in Form des *MorphiaConnectionController* konfiguriert werden. Weiters zeigt das Beispiel die Systematik, wie ein *CaseIdGenerator* sowie *Mockups* der *TraceHandler*-Instanz zugewiesen werden können. Wichtig ist zu beachten, dass die Eigenschaft *useJPA* der *Tracer-Bean* auf *false* gesetzt werden muss, um die Verwendung der eigenen Implementierung zu ermöglichen.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Configures the Camel Context-->
3
4 <beans ...>
5   <camelContext id="camel" xmlns="...">
6         <route>
7               ...
8         </route>
9   </camelContext>
10
```

```
11 <!-- configure our trace handler -->
12 <bean id="morphiaTraceEventHandler" class="MorphiaTraceEventHandler">
13    <!-- configure the morphia controller -->
14    <property name="morphiaController">
15       <bean class="MorphiaConnectionController">
16          <!-- configure the mongoDB connection information -->
17          <property name="mongoDBConnInfo">
18            <bean class="MongoDBConnectionInfo">
19               <property name="hostname" value="localhost"/>
20               <property name="port" value="27017"/>
21               <property name="database" value="traceDB"/>
22                  <!-- configure the mongoDB login credentials -->
23                  <property name="credentials">
24                     <bean class="MongoDBCredentials">
25                        <property name="username" value=""/>
26                        <property name="password" value=""/>
27                     </bean>
28                  </property>
29            </bean>
30          </property>
31       </bean>
32    </property>
33    <!-- configure the caseIdGenerator -->
34    <property name="caseIdGenerator">
35       <!--<bean class="SequenceCaseIdGenerator" />-->
36       <bean class="UUIDCaseIDGenerator" />
37    </property>
38    <!-- configure some mockups for testing purposes -->
39    <property name="mockups">
40       <list value-type="Mockup">
41  <bean class="ResourceMockup" />
42       </list>
43    </property>
44 </bean>
45
46 <!-- START SNIPPET: e1 -->
47 <!-- use camel jpa trace so we can see all -->
48 <!-- the traced exchanges in a database -->
49 <bean id="camelTracer" class="Tracer">
50    <!-- turn on jpa tracing, otherwise the -->
51    <!--TraceEventMessage is a non JPA Entity class -->
52    <property name="useJpa" value="false"/>
53       ...
```

```
54      <property name="traceHandler" ref="morphiaTraceEventHandler"/>
55    </bean>
56    <!-- END SNIPPET: e1 -->
57
58 </beans>
```

Listing 4.17: SpringXML Konfiguration des MorphiaCamelTracer

Das entwickelte Projekt *Morphia Camel Tracer* wurde veröffentlicht [1] und ist somit für weitere Verwendungen frei zugänglich. Das Entwicklungsresultat wurde auch in einem eigenen Apache Maven-Repository [2] zur Verfügung gestellt und kann somit in bestehende Apache Maven-Projekte eingebunden werden. Grundlagen zu Apache Maven [43] werden in dieser Diplomarbeit nicht im Detail erläutert, können jedoch auf der Projekthomepage oder in den vorhanden Dokumentationen nachgelesen werden.

4.4 Import- und Konvertierungsmechanismus

4.4.1 Einführung

Ziel der Diplomarbeit war es zu zeigen, wie ein Ereignisprotokoll zeitnah an Process Mining Techniken angebunden werden kann. Hierzu wurde die Entscheidung getroffen das in Kapitel 2 besprochene Process Mining Framework ProM als Zielumgebung für die Durchführung der Process Mining Techniken zu wählen. Wie in der Architektur beschrieben, wurde ein Import- und Konvertierungsmechanismus entwickelt, mit welchem die protokollierten Ereignisdaten in das XML Format XES [10] [11] [41] transformiert und in dieser Form in den Objekt-Pool von ProM geladen werden. In den folgenden Punkten wird das Format XES vorgestellt sowie der entwickelte Import- und Konvertierungsmechanismus erläutert und relevante Implementierungsdetails diskutiert.

4.4.2 XES - eXtensible Event Stream

Wie bereits im Kapitel 2 Process Mining erwähnt, können Ereignisprotokolle in unterschiedlichster Strukturierung und Gliederung auftreten. Grund hierfür ist unter anderem, dass in unterschiedlichen Quellsystemen die entsprechenden Ereignisprotokolle nicht nach einem einzigen gültigen Standard sondern aufgrund verschiedenen Konzepten und Ansätzen abgebildet werden. Das eXtensible Event Stream Format (XES) ist ein XML basierter Standard für die Darstellung von Ereignisprotokolldaten. Es bietet eine generische Darstellungsform von Ereignisprotokollen für den Austausch zwischen Analysetools (z.B.: ProM Framework) und der eigentlichen Applikationsdomäne, aus welcher das Protokoll generiert wurde. Wie im vorgeschlagenen Process Mining Vorgehensmodell beschrieben, ist es notwendig die Protokolldaten eines Informationssystems für die Analyse entsprechend aufzubereiten und bei Bedarf in ein entsprechendes Format zu konvertieren. Dies bedeutet im Falle der Nutzung von XES zur Darstellung der

[1] *http://morphia-camel-tracer.googlecode.com*
[2] *http://morphia.googlecode.com/svn/mavenrepo*

Protokolldaten, dass ein Ereignisprotokoll (z.B.: aus einer Collection einer dokumentbasierten Datenbank) mittels eines entsprechenden Protokolldatenkonverters in die XML basierte Darstellung von XES übergeführt werden muss. Solch ein Implementierungsansatz wird im Zuge der Vorstellung und Erläuterung der praktischen Umsetzung im Detail diskutiert.

Die Entwicklung des XML-basierten Standards XES verfolgte vier primäre Ziele:

- Einfache Darstellung und Abbildung von Ereignisprotokolldaten
- Flexibilität in der Darstellung von Ereignisprotokollen
- Hoher Grad an Erweiterbarkeit des Datenformats
- Ausdrucksstarke Repräsentation der Daten innerhalb des Formats

XES ist durch eine simple zielgerichtete Strukturierung der zentralen Elemente relativ einfach generierbar und durch die Abbildung als maschinenlesbares Format in Applikationen entsprechend gut einsetzbar. Ziel der Entwicklung war es, dass der Aufwand für die Überführung und Verwendung des XES-Formats relativ gering ist. Ein weiteres Schlüsselelement des XES-Formats ist die Flexibilität hinsichtlich der Quellsysteme. XES muss unabhängig vom jeweiligen Quellsystem generierbar sein, d.h. es darf keine bewusste enge Kopplung zu einen Quellsystem bestehen. Zusätzlich wurde im Zuge der Entwicklung darauf Acht gegeben, dass das Format für zukünftige Erweiterungen offen gestaltet wird. Einerseits muss bei der Erweiterung des Standards eine Vorwärts- sowie Rückwärtskompatibilität des Formats mit anderen Versionen bestehen, andererseits soll die Möglichkeit zu eigenen domänenspezifischen Erweiterungen des Protokolls möglich sein. Die in XES überführten Ereignisprotokolle sollen, im Vergleich zu dem ursprünglichen Ausgangsereignisprotokoll (z.B.: Dokumente aus einer dokumentbasierten Datenbank), so wenig Informationsverlust wie möglich aufweisen. XES erfordet es zwingend, alle Informationen des zu überführenden Ereignisprotokolls entsprechend zu typisieren, um den Informationsverlust bei der Transformation möglichst gering zu halten. Des Weiteren wird das Hinzufügen von semantischen Bedeutungen der abgebildeten Informationen, mittels einer generischen Methodik unterstützt.

XES-Struktur

Der Aufbau des XES-Formats lässt sich wie im abgebildeten Metamodell (Abbildung 4.13) darstellen. Folgend wird auf die Schlüsselelemente des XES-Metamodells im Detail eingangen und diese erläutert.

Die Basisstruktur eines XES-Dokuments besteht aus den Elementen *Log*, *Trace* und *Event*. Das *Log*-Element ist das Topelement innerhalb des XML-Dokuments und umfasst *n Trace*-Einträge sowie weitere Informationen zu den innerhalb des Dokuments verwendeten Erweiterungen, globale Attributdefinitionen sowie die Definition von Classifiern. Diese ermöglichen es, Attribute mit unterschiedlichen Bezeichnern zu einer identischen semantischen Bedeutung zusammenzufassen. Ein *Trace*, ein Unterelement einer *Log*-Struktur, ist von der Bedeutung äquivalent zu dem im bisherigen Dokument als Prozessinstanz (Case) beschriebenen Konstrukts, welches einen Prozessdurchlauf in einem Informationssystem abbildet. Ein *Trace* Element kann

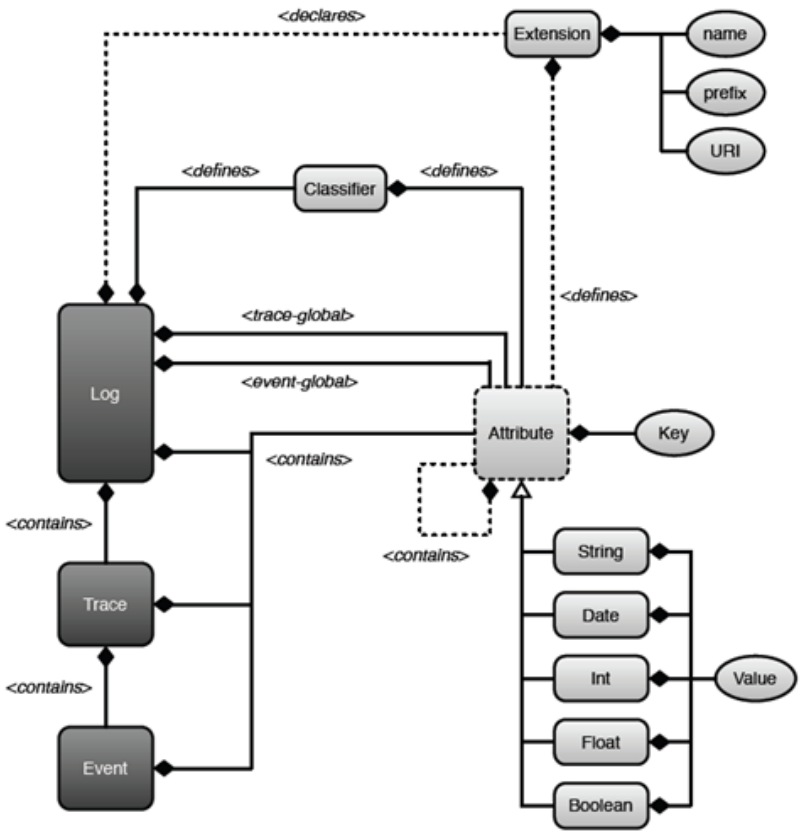

Abbildung 4.13: XES Metamodell

selbst wieder *m* verschiedene *Events* enthalten. Diese bilden die konkreten Aktionen bzw. Ereignisse innerhalb einer Prozessinstanz ab. Es ist zu beachten, dass die Anzahl der *Traces* innerhalb eines *Log* Elements sowie die Anzahl der Ereignisse innerhalb eines *Traces* beliebig groß sein können. Für gleichartige Prozessinstanzen kann die Anzahl der zugehörigen Ereignisse unterschiedlich sein, beispielsweise, wenn die Prozessinstanzen unterschiedliche Abläufe oder auch Spezialfälle abbilden. Die Elemente *Log*, *Trace* und *Event* enthalten selbst keine Informationen, sie werden lediglich dazu verwendet, die Struktur des Dokuments zu definieren. Alle Informationen welche im Ereignisprotokoll abgebildet werden sollen, müssen mit entsprechenden Attributen beschrieben werden.

Attribute in XES sind als ein Schlüssel-Wert Paar (wie in XML) abgebildet und werden zur Beschreibung des jeweilig zugehörigen Elternelements (z.B.: zu einem *Event*) genutzt. Die Werte der Attribute können, wie im XES-Metamodell illustriert, verschiedene Datentypen (z.B.: *String*, *Date*, *Int*, etc.) annehmen, wobe der Schlüssel des Attributes als Zeichenkette abgebildet ist. Um die Flexibilität innerhalb des Formats zu erhöhen, unterstützt XES die Definition von verschachtelten Attributen. Ein verschachteltes Attribut ist in XES als Unterelement eines Attributs realisiert. Dies ist notwendig um Informationen in einer effizienten Art und Weise zu

strukturieren, was es ermöglicht verschiedene Datentypen effizient zu einer Einheit zusammenzufassen. Um sicherzustellen, dass alle für die jeweiligen XES-Elemente (*Logs*, *Traces* sowie *Events*) zwingend notwendigen Attributinformationen verfügbar sind, wurde das Konzept von globalen Attribute im XES-Format eingeführt. Globale Attribute können entweder für die *Trace* oder die *Event* Ebene im XES-Ereignisprotokoll definiert werden. Diese Attribute beschreiben, welche Attributinformationen in jedem *Event* oder jedem *Trace* des Protokolls zwingend angeführt werden müssen.

Im Listing 4.18 ist ein gekürztes Beispiel eines im XES-Format abgebildeten Ereignisprotokolls illustriert. Zu beachten ist, dass nur Basisstrukturelemente inklusive der für das Beispiel notwendigen Attribute abgebildet sind. Aufgrund des zu großen Umfangs wurden etwa Informationen zu den Extensions und Classifiern nicht in das Dokument übernommen. Detailliertere Beispiele von XES sind im *open XES Developer Guide* [10] und in der *XES Standard Definition* [11] nachzuschlagen.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This file has been generated with the OpenXES library. -->
<!-- It conforms to the XML serialization of the XES standard -->
<!--  for log storage and management. -->
<!-- XES standard version: 1.0 -->
<!-- OpenXES library version: 1.0RC7 -->
<!-- OpenXES is available from http://www.openxes.org/ -->
<log xes.version="1.0" xes.features="nested-attributes"
          openxes.version="1.0RC7"
          xmlns="http://www.xes-standard.org/">

  <string key="source" value="Morphia TraceDB"/>
  <string key="concept:name" value="morphiaImport.xes"/>
  <string key="lifecycle:model" value="standard"/>
  <string key="description" value="Morphia Camel Trace"/>
  <trace>
    <string key="concept:name" value="3"/>
    <string key="description" value="Morphia trace instance"/>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
          <date key="time:timestamp" value="2011-03-11T11:16:13.204"/>
          <string key="concept:name" value="Event A"/>
          <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
          <date key="time:timestamp" value="2011-03-11T11:16:13.230"/>
          <string key="concept:name" value="Evemt B"/>
          <string key="lifecycle:transition" value="complete"/>
    </event>
```

```
31        ...
32    </trace>
33    <trace>
34      ...
35    </trace>
36      ...
37 </log>
```

Listing 4.18: XES: Beispiel-Ereignisprotokoll

Die bisher beschriebenen Konzepte des XES-Formats sind ausreichend, um die Struktur des Ereignisprotokolls und mittels der Attribute die Informationen dieser Strukturen zu beschreiben. Jedoch ist mit diesen Konzepten keine Möglichkeit zur Definition der Semantik dieser Attribute möglich. Für diesen Zweck steht in XES die *Extension*-Schnittstelle zur Verfügung, welche die Definition von Semantik erlaubt. Eine *Extension* in XES erlaubt es für ein Set an Attributen der unterschiedlichen Ebenen die Semantik zu definieren und stellt somit eine Referenz zur spezifischen Interpretation dieser Attribute zur Verfügung. Dies erlaubt es, semantische Information einem Attribut-Set zuzuführen. XES hat per Standard einige häufig benötigten *Extensions* definiert, jedoch macht es aus Sicht des Metamodells keinen Unterschied, ob es sich um eine Standard-Extension oder eine individuell definierte *Extension* handelt. Standard-*Extension* des XES-Formats sind:

- Concept – Attributinformation mit dem generell verständlichen Namen von Typhierarchie-Elementen
- Lifecycle – den Übergangszustand eines Event für die protokollierte Aktivität
- Organizational – ein Akteur, welcher beispielsweise für die Ausführung verantwortlich war
- Time – das exakte Datum vom jeweiligen Protokolleintrag
- Semantic - beschreibt abhängig von der Prozessbetrachtung die semantische Bedeutung
- Classification – klassifiziert Events aufgrund ihres logischen Konzepts

Überführung in XES

Um ein Ereignisprotokoll in das XES-Format überführen zu können, wird mit der *OpenXES JAVA Bibliothek* [49] eine entsprechende Schnittstelle (API) zur Verfügung gestellt. Das in dieser Bibliothek umgesetzte Klassenmodell ist analog zum beschriebenen XES-Metamodell umgesetzt worden, um beispielsweise die Entwicklung von notwendigen Ereignisprotokoll-Konverter zu erleichern. Äquivalent zum XES-Metamodell gibt es entsprechende Java-Klassen *XLog*, *XTrace* und *XEvent*.

Objektinstanzen dieser Klassen können durch Nutzung einer Factory erzeugt werden. Die erzeugten Objektinstanzen müssen in Folgen dem entsprechenden Elternelement über die bereitstehenden Zugriffsmethoden zugeordnet werden. Im Zuge der Umsetzung der die Diplomarbeit begleitenden praktischen Arbeit, wurde ein entsprechender Ereignisprotokoll-Konverter implementiert. Dieser Konverter wird im weiteren Verlauf der Arbeit, inklusive der relevanten Teile der *OpenXES API*, noch diskutiert, sowie Details der Implementierung beschrieben.

4.4.3 Import- und Konvertierungsmechanismus: Morphia Camel Tracer ProM Import PlugIn

Der zweite Implementierungsteil im Zuge dieser Diplomarbeit war die Entwicklung eines Importmechanismuses für das bestehende Process Mining Framework ProM. Es wurde eine Erweiterung des bestehenden ProM Framework umgesetzt, um das durch den *Morphia Camel Tracer* erzeugte Ereignisprotokoll im ProM Framework als Ausgangsbasis für Mining- oder Analyse-Algorithmen nutzen zu können. Der Mechanismus sollte im Speziellen eine zeitlich nähere Anbindung von bestehenden Process Mining Techniken an aktuelle Prozessdaten ermöglichen. Wie bereits am Anfang des Kapitels besprochen, werden innerhalb eines *Apache Camel*-Projekts Ereignisprotokolldaten erzeugt, welche den Mining- und Analyse-Algorithmen mit minimaler Vorbereitungs- und Verarbeitungszeit zur Verfügung gestellt werden sollen.

Die Überlegung war im Gegensatz zu bestehenden Lösungsansätzen, nicht aufgrund von extrahierten Datenständen (in Form von XES konvertierten Dateien) zu arbeiten, sondern die Datenquelle des Ereignisprotokolls in Form eines Import-PlugIns direkt im ProM Framework zugänglich zu machen. Dies erforderte die Entwicklung eines Import-PlugIns, welches es dem Benutzer ermöglicht den Zugriff auf ein vom *Morphia Camel Tracer* erzeugtes Ereignisprotokoll zu konfigurieren und den relevanten Ausschnitt der Prozessdaten in den Objekt-Pool von ProM zu laden. Um das Ereignisprotokoll aus der MongoDB laden zu können, müssen einerseits die Verbindungsparameter vom Benutzer einstellbar sein sowie Filter- und Importoptionen entsprechend der Anforderung gewählt werden können. Diese Parameter des PlugIns werden in den folgenden Punkten noch im Detail erklärt.

Zum Abschluss des Ereignisprotokolldatenimports, ist es erforderlich das Ereignisprotokoll in das XML-basierte Format XES zu überführen. Dies erfordert eine Konverter-Komponente, welche aufgrund von den gegebenen Parametern und einer Liste von *MorphiaTraceEventMessages*, ein Ereignisprotokoll im XES Format erzeugt. Detailierte Informationen zur Entwicklung von ProM-PlugIns können im verfügbaren *ProM Plugin Developer Guide* [51] oder in den entsprechenden Artikeln der Projektwebseite nachgelesen werden. Leider ist anzumerken, dass die vorhandenen Dokumente teilweise veraltet oder fehlerhaft sind. Die aktive ProM Community ist jedoch bei spezifischen Themen hilfsbereit und man kann durch Nutzung unterschiedlichen Kommunikationswege (z.B.: Mailing List, Forum, etc.) mit rascher Hilfe rechnen. Die wichtigsten Schlüsselelemente der entwickelten PlugIns setzen sich wie in Abbildung 4.14 gezeigt zusammen:

- **MorphiaTracePlugin:** Zentrale ProM-PlugIn-Klasse

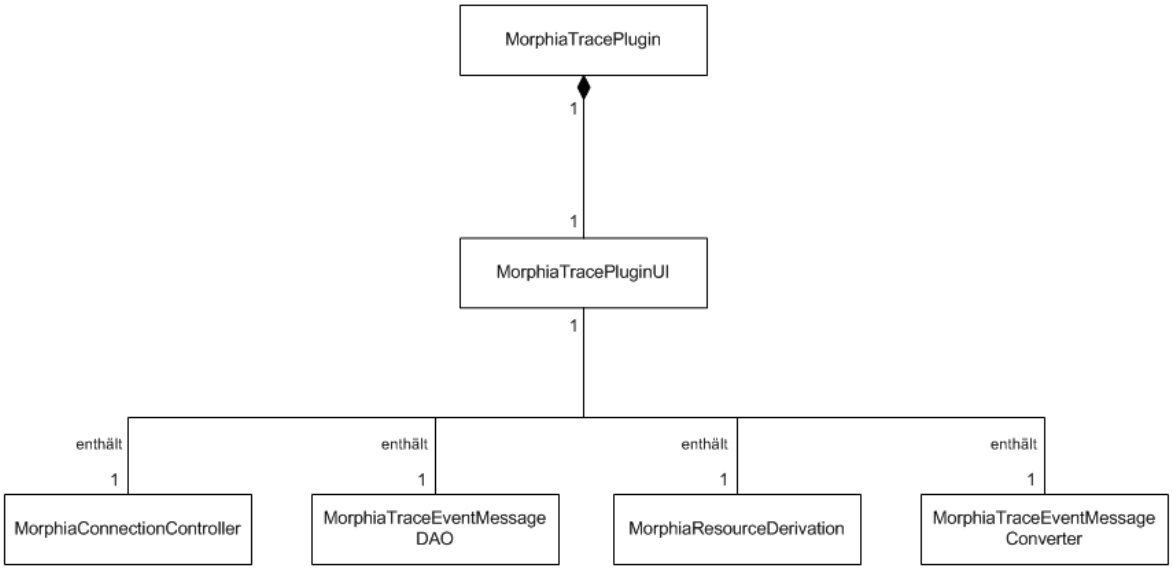

Abbildung 4.14: Klassendiagramm Morphia Camel Tracer ProM Import Plugin

- **MorphiaTracePluginUI:** Benutzeroberfläche des PlugIns inklusive Steuerungslogik
- **MorphiaConnectionController:** Klasse zur Verwaltung der Morphia-Verbindung zur MongoDB (wie im Teilkapitel Protokollierungsmechanismus beschrieben)
- **MorphiaResourceDerivation:** Ableitungslogik der Akteurinformation der Ereignisse bzw. Prozessinstanzen
- **MorphiaTraceEventMessageConverter:** Konvertierungslogik zur Überführung des Ereignisprotokolls in das XES Format

Diese Schlüsselelemente der Implementierung sowie die getroffenen Architekturentscheidungen, werden in den folgenden Punkten dieses Kapitels beleuchtet. Des Weiteren wird im Zuge dieser Erläuterungen der Ablauf der PlugIns im Allgemeinen und der Ablauf des Konvertierungsmechanismus im Speziellen erklärt.

MorphiaTracePlugin

Zentraler Bestandteil eines ProM PlugIns (in der Framework Version 6.0) ist eine, entsprechend der ProM Developer-Anleitung zur Entwicklung von PlugIns, entwickelte Java Klasse. In der entwickelten Lösung wurde diese Klasse (*MorphiaTracePlugin*) als zentrales Element der PlugIn Lösung entwickelt. Zwei Charakteristiken muss eine ProM-PlugIn-Klasse zumindest erfüllen: a) Die notwendigen Annotationen müssen in der Klasse definiert werden und b) es muss eine *main*-Methode implementiert sein. In Listing 4.20 ist der Aufbau solch einer Klasse dargestellt.

Die Klasse muss die Annotation *@Plugin* enthalten, mit welcher der Name des PlugIns, die Bezeichnung des PlugIn-Ergebnisses (in unserem Fall das Ereignisprotokoll) sowie der Objekttyp des PlugIn-Ergebnisses deklarativ beschrieben werden. Die zweite Annotation *@UITopiaVariant* konfiguriert die Zugehörigkeit des PlugIns und fügt Meta-Informationen über den Entwickler des PlugIns bei, welche in Folge im ProM Framwork *UITopia* bei der PlugIn-Auswahl angezeigt werden. Weiters muss eine *static main*-Methode implementiert werden, für welche der Rückgabewert mit der Angabe der *@PlugIn* Annotation übereinstimmen muss.

Für das *MorphiaTracePlugin* ist dies beispielsweise ein Objekt des Typs *XLog*, welches das Ereignisprotokoll im XES-Format darstellt. Als Parameter erhält die Methode ein *Context*-Objekt des aufrufenden Frameworks (in unserem Fall ProM UITopia), welches als UI-Container für die Benutzeroberfläche verwendet werden kann. Die *main*-Methode enthält selbst nur den Aufruf der jeweiligen Benutzeroberfläche sowie die Rückgabe des entsprechenden Rückgabeobjekts des Typs *XLog*. Durch Aufruf der Methode *context.getFutureResult(0).setLabel("... ")* ist es möglich dem Objekt-Pool eine Anzeigebeschreibung für das Rückgabeobjekt mitzuteilen.

```
1  import org.deckfour.xes.model.XLog;
2  import org.processmining.contexts.uitopia.UIPluginContext;
3  import org.processmining.contexts.uitopia.annotations.UITopiaVariant;
4  import org.processmining.framework.plugin.annotations.Plugin;
5  import org.processmining.framework.plugin.annotations.PluginVariant;
6  import org.processmining.plugins.ui.MorphiaTracePluginUI;
7
8  @Plugin(name = "Morphia␣CamelTrace␣Importer",
9                 parameterLabels = {  },
10                returnLabels = { "Log" },
11                returnTypes = { XLog.class },
12                userAccessible = true)
13
14 public class MorphiaTracePlugin {
15
16    @UITopiaVariant(affiliation = UITopiaVariant.EHV ,
17                            author = "Thomas␣Neuboeck",
18                            email = "thomas.neuboeck@infor.com")
19
20    public static XLog main(final UIPluginContext context) {
21       MorphiaTracePluginUI filterUI =
22           new MorphiaTracePluginUI(context);
23       XLog log = filterUI.load();
24       context.getFutureResult(0).setLabel("MorphiaCamelTraceImport");
25       return log;
26    }
27
```

```
28 }
```

Listing 4.19: MorphiaTracePlugin - ProM-Import-PlugIn

MorphiaTracePluginUI

MorphiaTracePluginUI stellt die Benutzeroberfläche und die darin enthaltene Steuerungslogik des Import-PlugIns zur Verfügung. Die Benutzeroberfläche wurde im Zuge des Entwurfs in drei Teilschritte unterteilt:

- Verbindungskonfiguration der Datenquelle
- Konfigurationeinstellungen des Datenimports
- Filtereinstellungen für das Ereignisprotokoll

In der Abbildung 4.15 ist die Klassenstruktur der umgesetzten Benutzeroberfläche dargestellt, welche an den strukturellen Aufbau bestehender PlugIn-Entwicklungen angelehnt wurde. Der Steuerungsablauf des PlugIns wurde als sequentieller Ablauf der beschriebenen Teilschritte konzipiert (siehe Abbildung 4.16). Jedoch erlaubt die implementierte Steuerungslogik dem Benutzer, zu vorhergehenden Konfigurations- oder Einstellungsschritten zurück zu springen oder den gesamten Importprozess abzubrechen. *MorphiaTracePluginUI* ist die zentrale Klasse der Benutzeroberfläche und enthält eine Liste an Teilschritten (=Benuterzoberflächen) vom Typ *MorphiaTracePlugInAbstractStep*. Weiters übernimmt diese Klasse die Aufgabe der Steuerungslogik zwischen den einzelnen Teilschritten.

Die abstrakte Klasse *MorphiaTracePlugInAbstractStep* ist eine Subklasse von *javax.swing.JPanel*, welche dem *UIContext*-Objekt des ProM Framework zur Anzeige übergeben werden kann (siehe Listing 4.20). Um allgemeines Verhalten sowie Variablen abzubilden, wurde noch eine weitere Klasse *MorphiaTracePlugInSimpleStep* eingeführt, welche von der abstrakten Klasse *MorphiaTracePlugInAbstractStep* erbt und das allgemein gültige Verhalten der Benutzeroberfläche umsetzt. Ein Teilschritt erbt in Folge von der Klasse *MorphiaTracePlugInSimpleStep* und implementiert die jeweilige individuelle Benutzeroberfläche sowie -interaktion. Listing 4.20 zeigt die notwendigen Ausschnitte zur Übergabe der Benutzeroberfläche (Ableitung von *JPanel*) zur Anzeige an den *UIContext* sowie zur Verarbeitung des Interaktionsergebnisses der Benutzeroberfläche (z.B.: NEXT oder FINISHED).

```
1 import org.deckfour.uitopia.api.event.TaskListener.InteractionResult;
2 ...
3 result = context.showWizard("Description",
4             currentStep == 0,
5             currentStep == nofSteps - 1,
6             mySteps[currentStep]);
7 ...
8 switch (result) {
```

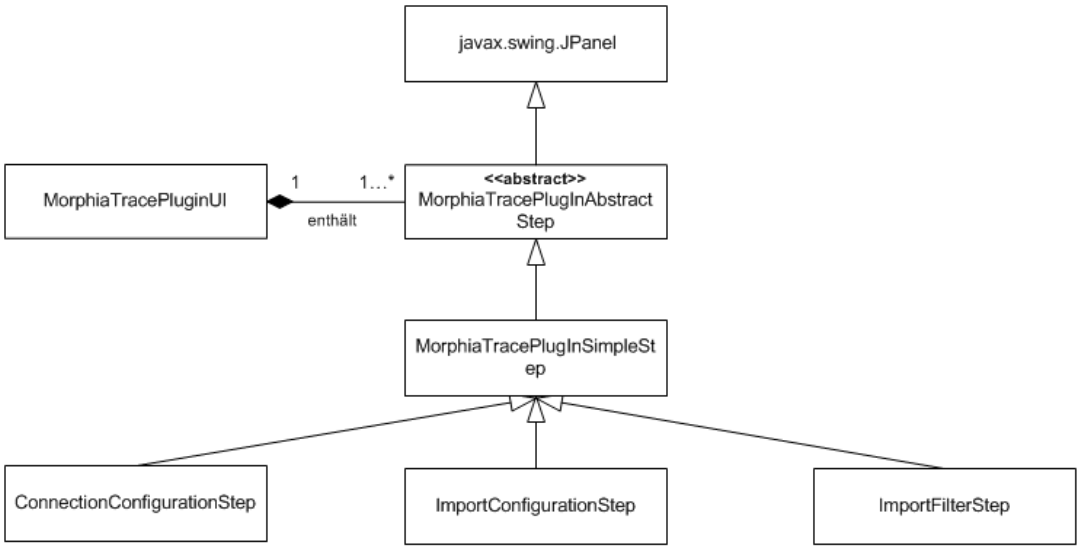

Abbildung 4.15: Klassendiagramm Morphia Camel Tracer ProM Import Plugin UI

```
9       case NEXT :
10      ...
11      case PREV :
12      ...
13      case FINISHED :
14      ...
15  }
```

Listing 4.20: MorphiaTracePlugin - ProM Import PlugIn UI Interaktion mit ProM UI Context

Die Verbindungskonfiguration (siehe Abbildung 4.17) erlaubt es dem Benutzer die Konfiguration der Zugriffsparameter auf das gewünschte Ereignisprotokoll innerhalb einer MongoDB-Instanz durchzuführen. Analog zum *Morphia Camel Tracer* wird hierzu das ODM Framework Morphia für den Datenzugriff genutzt. Die ersten Überlegungen waren, die Verbindungskonfiguration über eine explizite Konfigurationsdatei zu erlauben. Um dem PlugIn einen höheren Flexibilitätsgrad zu verleihen, fiel die Entscheidung auf die Implementierung einer Konfigurationsmöglichkeit innerhalb des Import PlugIns. Der Benutzer muss daher im ersten Schritt den Hostnamen sowie den Port für die Datenbankinstanz der MongoDB konfigurieren. Falls die Instanz eine Authentifizierung erfordert, kann Benutzername und Passwort durch die Aktivierung des *SecureMode* eingegeben werden.

Sobald die notwendigen Parameter eingegeben wurden, kann mittels der Schaltfläche *Connect* die Verbindung zur Datenbank hergestellt werden. Im Falle eines Fehlschlags beim Aufbau der Datenbankverbindung, wird dem Benutzer eine entsprechende Meldung ausgegeben. Im Hintergrund nutzt die Oberfläche den, bereits im vorhergehenden Kapitel beschriebenen, *Morphia-*

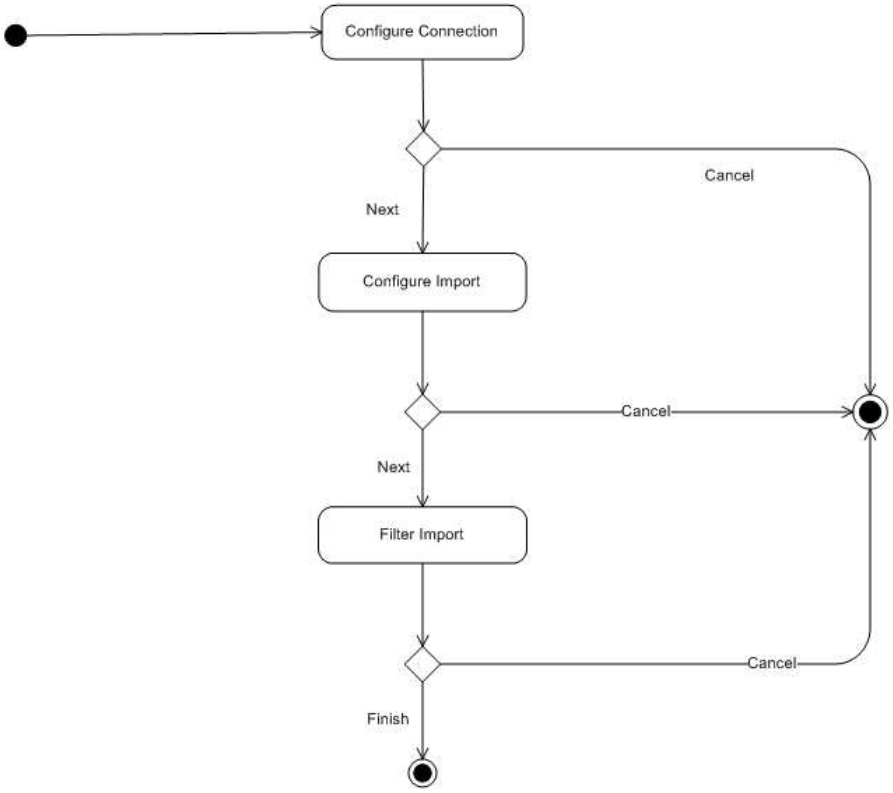

Abbildung 4.16: Ablaufdiagramm Morphia Camel Tracer ProM Import Plugin

ConnectionController um die Verbindungsdaten zu verwalten und die notwendigen Objekte für den Datenzugriff zu erzeugen. Nach einem erfolgreichen Verbindungsversuch wird dem Benutzer die Liste an verfügbaren Datenbanken (wobei nicht verifiziert wird ob diese ein gültiges Ereignisprotokoll enthalten) zur angezeigt. Nach Auswahl der gewünschten Datenbank kann mittels der Schaltfläche *Next* der nächste Teilschritt aufgerufen werden.

Im nächsten Teilschritt, den Konfigurationseinstellungen des Datenimportes (Abbildung 4.18), steht dem Benutzer die Möglichkeit offen, spezifische Konfigurationen für den Import vorzunehmen. In der entwickelten Version des PlugIns, kann die Variante, wie die Information zum Akteur (in der Benutzeroberfläche wird dies als Ressource bezeichnet) aus den Ereignisprotokolldaten ermittelt werden soll, vom Benutzer ausgewählt werden. Im Zuge der Entwicklung des *Morphia Camel Tracer* hat sich gezeigt, dass das Quellinformationssystem in der aktuellen Version, noch keine Akteurinformationen zu den einzelnen Ereignissen zur Verfügung stellt.

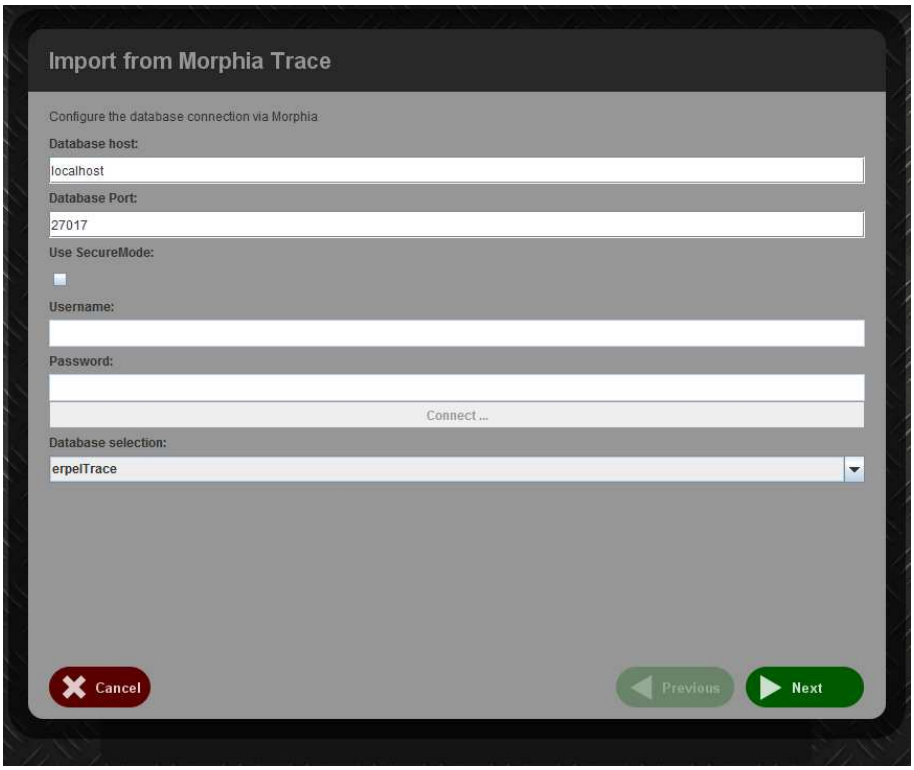

Abbildung 4.17: ProM-Import-PlugIn: Benutzeroberfläche zur Konfiguration der Verbindung zur MongoDB

Eine Erweiterung des Informationssystems, um solch eine Information für gewisse Ereignisse der Prozesse bereitzustellen, ist in Zukunft potentiell denkbar. Aufgrund dieser Möglichkeit und dem Wunsch in Zukunft auch Modelle der organisatorischen Perspektive ableiten zu können, wurde entschieden, im vom Import-PlugIn extrahierten und konvertierten Ereignisprotokoll die Information zum Akteur mitzuführen. Folglich wurde eine entsprechende Ableitungslogik konzipiert (diese wird im Anschluss besprochen), welche unterschiedliche Ableitungsvarianten zur Verfügung stellt. Diese Ableitungslogiken extrahieren die vorhandene Akteurinformation aufgrund der getroffenen Konfiguration und bereiten diese im Ereignisprotokoll entsprechend auf.

Die über die Benutzeroberfläche verfügbaren Konfigurationseinstellungen ermöglichen es dem Benutzer, die jeweilig benötigte Variante zu wählen und, wenn notwendig, einen entsprechenden Standard-Wert sowie einen eindeutigen Schlüssel zur Ermittlung der Akteurinformation aus dem Header des Ereignisses zu definieren. Mit Bestätigung der Schaltfläche *Next* kann der Benutzer zum letzten Teilschritt übergehen.

Im letzten Schritt des PlugIns *Filtereinstellungen des Ereignisprotokolls* (Abbildung 4.19)

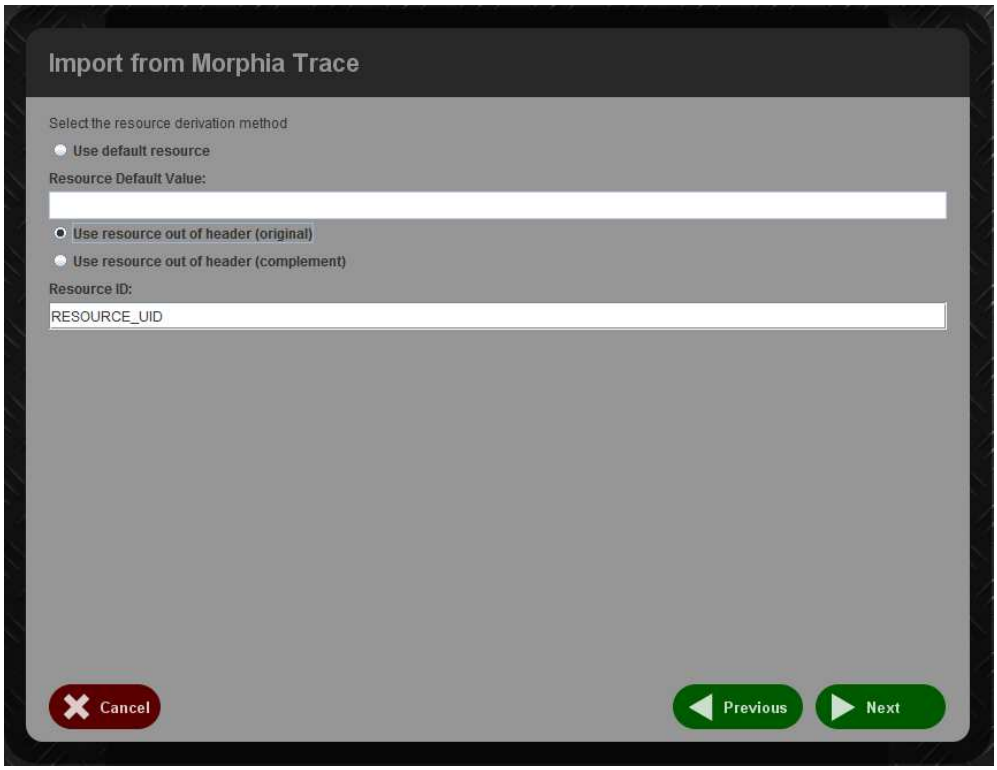

Abbildung 4.18: ProM-Import-PlugIn: Benutzeroberfläche zur Konfiguration des Imports

ist es dem Benutzer möglich entsprechende Einschränkungen der Ereignsprotokolldaten vor dem Import zu treffen. Umgesetzt wurde ein Filter aufgrund eines Zeitraums, welchen der Benutzer innerhalb der Oberfläche frei definieren kann. Soll keine Einschränkung getroffen werden, kann die Option *No Filter* gewählt werden. Unterstützt wird der Benutzer bei der Definition der Filterparameter durch die Anzeige der Anzahl von Ereignissen, welche aus den gewählten Filtereinstellungen resultieren. Mit der Schaltfläche *Finish* werden alle gewählten Einstellungen und Konfigurationen bestätigt und folglich dem *MorphiaTraceEventMessageConverter* (dieser wird im Anschluss beschrieben) übergeben.

MorphiaResourceDerivation

Wie bereits erwähnt wurde der Mechansimus *MorphiaResourceDerivation* zur Ermittlung und Ableitung der Akteurinformation aus den verfügbaren Ereignissen umgesetzt. Jener Mechanismus unterstützt drei Varianten zur Informationsermittlung und Aufbereitung derselbigen im extrahierten Protokoll.

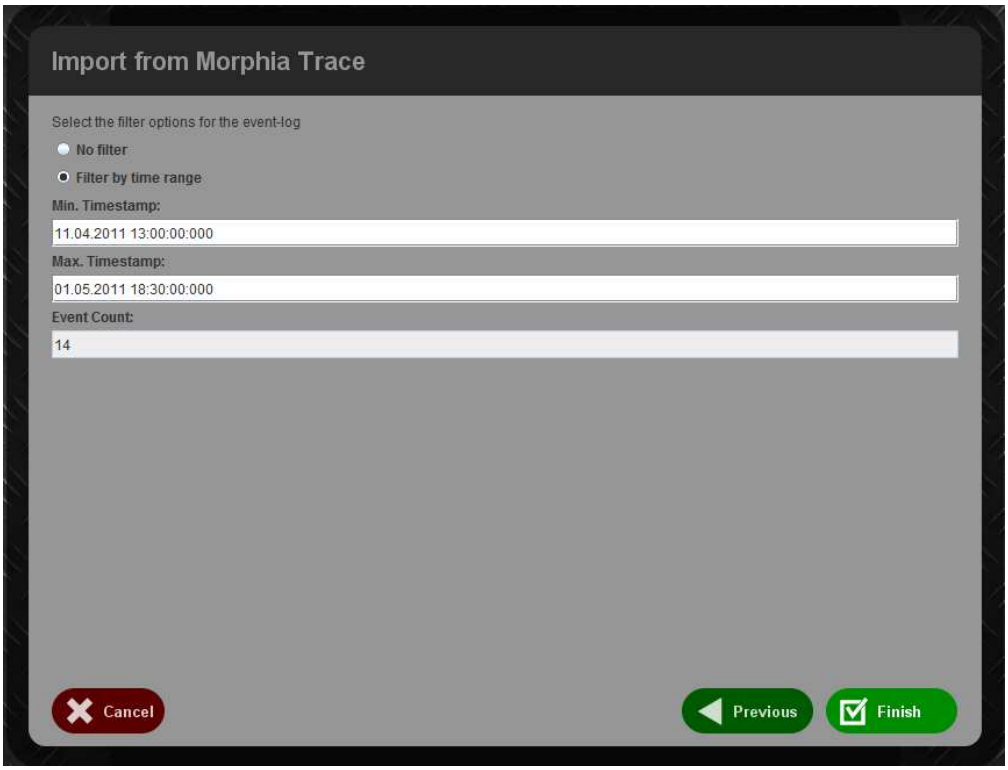

Abbildung 4.19: ProM-Import-PlugIn: Benutzeroberfläche zur Definition der Filtereinstellungen

- Fall A: Verwendung einer Default-Ressource
- Fall B: Verwendung der Akteurinformation aus dem Header einer Message
- Fall C: Verwendung der Akteurinformation aus dem Header einer Message mit Ergänzung

Fall A stellt die einfachste Variante dar, bei welcher der Benutzer eine Default-Ressource definiert, die allen Ereignissen der Prozessinstanz in Folge zugeteilt wird. Im Fall B wird davon ausgegangen, dass im Header einer *MorphiaTraceEventMessage* die Akteurinformation vorhanden ist. Der Benutzer kann in diesem Fall angeben unter welchem Schlüssel die Information im Header zu finden ist. Wird im Zuge der Konvertierung des Ereignisprotokolls eine Information zum angegebenen Schlüssel gefunden, dann wird diese als Akteurinformation verwendet. Wenn nicht, dann wird dem entsprechenden Attribut des Ereignisses eine leere Akteurinformation zugewiesen. Im Fall C wird ebenfalls von der Existenz der Akteurinformation im Header der *Message* ausgegangen. Jedoch erwartet man keine vollständige Existenz der Information (d.h. in allen Ereignissen der Prozessinstanz), sondern geht nur von einem partiellen Aufkommen (d.h.

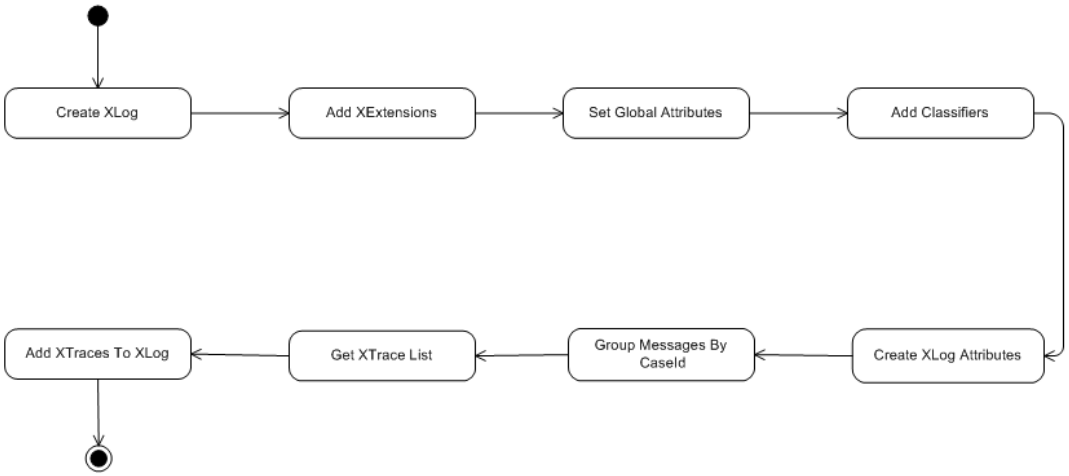

Abbildung 4.20: Ablauf MorphiaTraceEventMessageConverter

nur ein Set von Ereignissen) der Information aus.

Dieses Vorgehen soll Szenarien unterstützen, bei welchen nicht jedes Ereignis die Header-Information enthält (z.B.: die ersten Ereignisse einer Prozessinstanz), jedoch die selbe Akteurinformation wie die folgenden erhalten können oder sollen. In diesem Fall wird beim Konvertieren die erst gefundene Akteurinformation allen Ereignissen ohne Akteurinformation zugewiesen.

MorphiaTraceEventMessageConverter

Wie bereits beschrieben wird nach Abschluss der Importkonfiguration der Konverter mit den entsprechenden Informationen aufgerufen. Zum einen werden die Einstellungen zur Ableitung der Akteurinformationen als auch eine bereits gefilterte Liste von *MorphiaTraceEventMessages* übergeben. Die Aufgabe des Konverters *MorphiaTraceEventMessageConverter* besteht darin diese Liste in das bereits beschriebene Format XES zu überführen. Das Ergebnis des Konverters ist in Folge ein *XLog*-Objekt, welches dem *MorphiaTracePlugin* zur Rückgabe an das ProM Framework zur Verfügung gestellt wird. Der interne Ablauf des Konverters (siehe Abbildung 4.20) umfasst zwei Phasen:

- Vorbereitung der Struktur des XES-Ereignisprotokolls
- Überführung der Ereignisprotokolldaten in das XES-Format

Vorbereitung der XES-Struktur

Die Vorbereitung der XES-Struktur (siehe Abbildung 4.21) umfasst die Erstellung eines *XLog* Objekts sowie die Definition von zu verwendenden Extensions sowie Classifiern und die Festlegung von Attributen auf globaler sowie *XLog*-spezifischer Stufe. Die Überführung des Ereignisprotokolls in XES durchläuft einerseits die Ermittlung aller gültigen Prozessinstanz-IDs

sowie die Gruppierung der Ereignisse nach diesen. Im Ausgangsprotokoll liegen die Ereignisdaten (in Form von einzelnen Dokumente) in sequentieller Form vor. Die Überführung dieser Ereignissequenz hat die Aufgabe, die protokollierten Ereignisse in eine gruppierte Darstellung (nach Prozessinstanzen) zu transformieren.

In Folge wird auf Basis dieser Gruppierung die Instanzierung und Befüllung der internen Strukturen (*XTrace* und *XEvent*) durchgeführt. Diese Schritte beinhalten, analog wie zur ersten Phase, die Festlegung der Attribute auf Ebene der Prozessinstanz sowie des jeweiligen Ereignisses. Im Zuge der Befüllung der Ereignisdaten wird aufgrund der getroffenen Konfiguration, im Bedarfsfall, die Akteurinformation für die entsprechenden Ereignisse abgeleitet. Ist die Variante Fall C - Verwendung der Akteurinformation aus dem Header einer *Message* mit Ergänzung konfiguriert - muss die Logik nach Ermittlung aller Ereignisse der gesamten Prozessinstanz abschließend nochmals zur Ergänzung fehlender Akteurinformationen durchlaufen.

Überführung der Ereignisprotokolldaten
Um die vorliegenden Protokolldaten in Form von *MorphiaTraceEventMessages* in das XES-Format zu überzuführen war die Definition einer Transformationslogik, wie in Tabelle 4.6 dargestellt, notwendig.

Stufe	**Ziel**	**Quelle**	**Anmerkung**
XTrace	concept:name	CaseId	
XEvent	concept:name	ToNode	Start-Ereignis
XEvent	concept:name	PreviousNode	Complete-Ereignis
XEvent	time:timestamp	Timestamp	
XEvent	org:resource	[variabel]	siehe ResourceDerivation
XEvent	lifecycle:transition	'Start' / 'Complete'	

Tabelle 4.6: Konvertierungmechanismus: Transformation von MorphiaTraceEventMessage (nur relevante) in XES

Im Zuge der Definition der Überführungsregeln, wurde auf die explizite Auswahl von unbedingt notwendigen Informationen zur Darstellung im resultierenden Ereignisprotokoll geachtet. Ziel war es, das resultierende XES-Ereignisprotokoll so kompakt als möglich zu generieren, um weiterführenden Verarbeitungsschritten (z.B.: Mining Algorithmen) keinen unnötigen Ballast zu übergeben und folglich den Aufwand zur Durchführung zu entlasten. Die Ermittlung der Akteurinformation, ist in der Transformationstabelle nicht im Detail abgebildet. Diese ist aufgrund der getroffenen Einstellungen, wie im vorhergehenden Punkt *MorphiaResourceDerivation* beschrieben, durchzuführen.

Für die Ableitung des Attributs *lifecycle:transition* der abgebildeten Ereignisse, wurde die Implementierung einer simplen Logik gewählt. In diesem Attribut wird der zum Zeitpunkt des Übergangs relevante Zustand für das Ereignis abgebildet. Die Konvertierungslogik berücksichtigt zwei verschiedene Zustände: a) Start - Beginn der Aktivität und b) Complete - Abschluss der Aktivität. Im durch den *MorphiaEventTracer* erzeugten Ereignisprotokoll, kann diese Zu-

standsinformation des Ereignisse aufgrund der Eigenschaften *toNode* und *previousNode* abgeleitet werden. Da in diesem Ereignisprotokoll (wie im Zuge der Erläuterung des *Apache Camel Tracer*-Mechanismus beschrieben) die Übergänge zwischen den Aktivitäten erfasst sind, muss man zur Ableitung beider Zustände eines Ereignisses zwei Ereignisprotokolleinträge betrachten: a) den Übergang vor der Aktivität und b) den Übergang nach der Aktivität.

Im Laufe der Analyse der erzeugten Ereignisprotokolldaten zeigte sich, dass für das letzte Ereignis einer Prozessinstanz keine Information zum Complete-Ereignis aus den Daten abgeleitet werden kann. Grund hierfür ist, dass der *Apache Camel Tracer*-Mechanismus nach der Durchführung der letzten Aktivität keinen weiteren Übergang abfangen und folglich protokollieren kann (d.h. im Protokoll ist kein Eintrag mit *previousNode = 'letzte Aktivität'* und *toNode = null* vorhanden). Um aufgrund dieser Grundlage ein gültiges Protokoll im XES-Format zu erzeugen, war es notwendig, folgende Themen zu lösen:

- Identifizierung des letzten Ereignisses einer Prozessinstanz
- Darstellung des letzten Ereignisses einer Prozessinstanz im XES Format

Identifizierung des letzten Ereignisses
Für die Identifizierung des letzten Ereignisses, wurde die Annahme getroffen, dass alle Ereignisse, exklusive dem letzten Ereignis, in jeweils einem protokollierten Übergang in den Eigenschaften *toNode* sowie *previousNode* festgehalten wurden (Anmerkung: Übergänge zu den ersten Ereignissen enthalten eine Information zur *toNode*, aber keine zur *previousNode*). Aufgrunddessen kann somit eindeutig ein Ereignis aus dem Protokoll extrahiert werden, für welche diese Charakteristik nicht zutrifft.

Darstellung des letzten Ereignisses
Zur Darstellung des letzten Ereignisses im XES Format waren zwei Varianten möglich:

- Darstellung erfolgt in Form eines Complete-Ereignis ohne Start-Ereignis
- Ereignis wird in äquivalenter Art zu den anderen Ereignissen (d.h. Start- und Complete-Ereignis)

Vorteil einer äquivalenten Darstellung zu den restlichen Ereignissen ist eine einheitliche Repräsentation aller Ereignisse im gesamten XES Ereignisprotokoll. Nachteil dieser Variante ist die künstliche Erzeugung einer zweiten Ereignisinformation (da alle Attribute ident sind), welche keinen Informationsmehrwert für das konvertierte Protokoll bringt. In beiden Darstellungsvarianten ist es nicht möglich die Ereignisdauer für das letzte Ereignis zu ermitteln.

Für die implementierte Konvertierungslogik wurde eine äquivalente Darstellung berücksichtigt, da aus unserer Sicht das künstlich erzeugte Complete-Ereignis in weiterer Folge unproblematisch ist und eine homogene Abbildung der Ereignisinformationen gewährleistet. Das Resultat der umgesetzten Konvertierungslogik, ist ein in das XES-Format überführtes Ereignisprotokoll aus einer MongoDB-Quelle. Dieses Protokoll kann in weiterer Folge im ProM Framework

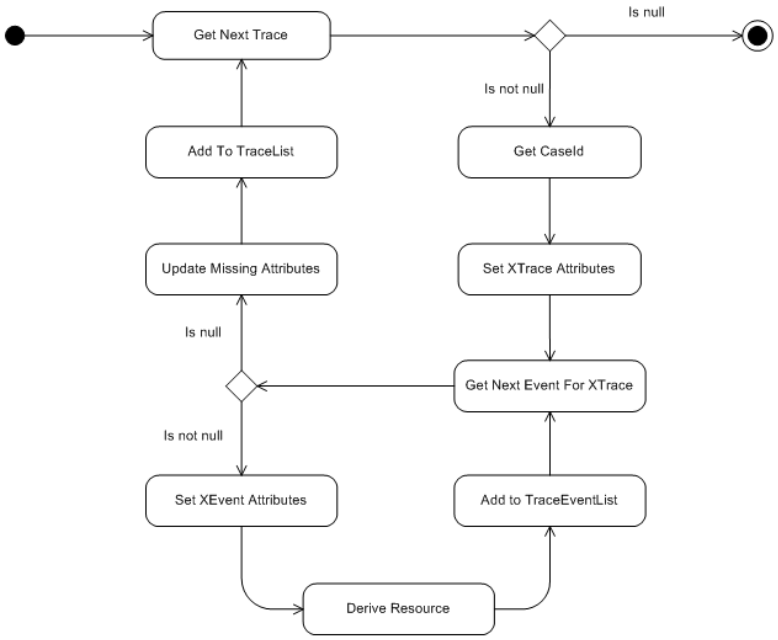

Abbildung 4.21: Ablauf: Überführung der Liste von MorphiaTraceEventMessages in XES

für Mining- und Analyse-Algorithmen als Eingabeinformation weiterverwendet werden.

Beispiel zur Transformation
Anhand des folgenden Beispielereignisprotokolls ist aufgrund der definierten Transformationsregeln die Überführung von Ereignisprotokolldaten in das XES-Format illustriert. In der Tabelle 4.7 sind Protokolldaten abgebildet, wobei jeweils eine Zeile als Repräsentation eines *MorphiaTraceEventMessage*-Dokuments zu sehen ist. Die dargestellten Daten wurden auf die Information zur Prozessinstanz (CaseID), den Informationen zum Vorgänger und Nachfolger (*PrevNode* und *ToNode*) sowie auf den Zeitstempel reduziert. Alle weiteren beschriebenen Eigenschaften der Dokumente sind für die Darstellung im Beispiel nicht relevant und wurden daher weggelassen. Die abgebildeten vier Beispieldokumente umfassen Einträge für die beiden Prozessinstanzen *C1* und *C2* mit den jeweiligen Ereignissen *E1*, *E4* und *E5*. Alle Ereignisse der Prozessinstanzen, welche zwischen den abgebildeten protokolliert wurden, sind nicht berücksichtigt. In den abgebildeten Ereignisprotokolldaten 4.7 sind für die letzten Aktivitäten aller Prozessinstanzen keine abschließenden Einträge (*previousNode = 'letzte Aktivität'* und *toNode = null*) vorhanden.

CaseID	PrevNode	ToNode	Timestamp	weitere Eigenschaften
C1	null	E1	2011-05-05 07:33:12	...
C2	null	E1	2011-05-05 13:49:44	...
...	...	...	...	...
C2	E4	E5	2011-05-05 14:15:32	...
C1	E4	E5	2011-05-05 14:18:05	...

Tabelle 4.7: Konvertierungmechanismus: Beispiel Ereignisprotokolldaten

Der Konverter muss, wie beschrieben, aufgrund einer sequenziellen Ausgangsbasis (pro Ereignis ein Dokument) die Daten in die XES-Darstellung überführen. Hierzu ist es notwendig, die unterschiedlichen Prozessinstanzen zu ermitteln und in Folge die vorliegenden Ereignisse aufgrund der ermittelten Prozessinstanzen zu gruppieren. Zu beachten ist, dass die Ereignisse unterschiedlicher Prozessinstanzen innerhalb der Ereignisliste vermischt vorliegen können (da die Prozessinstanzen parallel ausgeführt werden und sich über einen längeren Zeitraum erstrecken können). Das vom Konverter überführte Ereignisprotokoll ist im Listing 4.21 als Darstellung im XES-Format abgebildet. Der Inhalt des Listings ist ebenfalls auf die für das Beispiel relevanten Teile des resultierenden XES-Protokolls reduziert.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <log xes.version="1.0" xes.features="nested-attributes"
3           openxes.version="1.0RC7"
4           xmlns="http://www.xes-standard.org/">
5
6    ...
7    <trace>
8      <string key="concept:name" value="C1"/>
9      <event>
10          <date key="time:timestamp" value="2011-05-05T07:33:12.000"/>
11          <string key="concept:name" value="E1"/>
12          <string key="lifecycle:transition" value="start"/>
13            ...
14     </event>
15     <event>
16          <date key="time:timestamp" value="2011-05-05T14:18:05.000"/>
17          <string key="concept:name" value="E1"/>
18          <string key="lifecycle:transition" value="complete"/>
19            ...
20     </event>
21       ...
22     <event>
23          <date key="time:timestamp" value="2011-05-05T14:18:05.000"/>
24          <string key="concept:name" value="E4"/>
```

```
25          <string key="lifecycle:transition" value="complete"/>
26            ...
27      </event>
28      <event>
29          <date key="time:timestamp" value="2011-05-05T14:18:05.000"/>
30          <string key="concept:name" value="E5"/>
31          <string key="lifecycle:transition" value="start"/>
32            ...
33      </event>
34      <event>
35          <date key="time:timestamp" value="2011-05-05T14:18:05.000"/>
36          <string key="concept:name" value="E5"/>
37          <string key="lifecycle:transition" value="complete"/>
38            ...
39      </event>
40  </trace>
41  <trace>
42      <string key="concept:name" value="C1"/>
43      <event>
44          <date key="time:timestamp" value="2011-05-05T13:49:44.000"/>
45          <string key="concept:name" value="E1"/>
46          <string key="lifecycle:transition" value="start"/>
47            ...
48      </event>
49      <event>
50          <date key="time:timestamp" value="2011-05-05T14:15:32.000"/>
51          <string key="concept:name" value="E1"/>
52          <string key="lifecycle:transition" value="complete"/>
53            ...
54      </event>
55       ...
56      <event>
57          <date key="time:timestamp" value="2011-05-05T14:15:32.000"/>
58          <string key="concept:name" value="E4"/>
59          <string key="lifecycle:transition" value="complete"/>
60      </event>
61      <event>
62          <date key="time:timestamp" value="2011-05-05T14:15:32.000"/>
63          <string key="concept:name" value="E5"/>
64          <string key="lifecycle:transition" value="start"/>
65            ...
66      </event>
67      <event>
```

```
68            <date key="time:timestamp" value="2011-05-05T14:15:32.000"/>
69            <string key="concept:name" value="E5"/>
70            <string key="lifecycle:transition" value="complete"/>
71        </event>
72    </trace>
73    ...
74 </log>
```

Listing 4.21: Konvertierungmechanismus: Beispiel XES-Ereignisprotokoll

KAPITEL 5

Demonstration und Evaluierung der zeitnahen Integration von Ereignisprotokollen

5.1 Allgemein

In diesem Kapitel soll die Funktionalität des Process Mining Frameworks ProM (Version 6) anhand eines Beispiels demonstriert und evaluiert werden. Für diesen Zweck wurde eine simple *Apache Camel*-Beispielanwendung entwickelt, welche den beschriebenen Protokollierungsmechanismus zur Erzeugung von Ereignisprotokolldaten nutzt. Diese Prozessdaten wurden mittels des vorgestellten Import- und Konvertierungsmechanismus in das XES-Format übersetzt und in den Objekt-Pool des ProM Frameworks geladen. Für die Demonstration von Mining Algorithmen im ProM Framework wurden folgende Perspektiven berücksichtigt:

- Prozess Perspektive
- Organisatorische Perspektive

In den folgenden Absätzen wird die verwendete Beispielanwendung erklärt, sowie die Ergebnisse und Erkenntnisse der Durchführung aller Mining-Algorithmen in den unterschiedlichen, berücksichtigten Perspektiven präsentiert.

5.2 Apache Camel-Beispielanwendung

5.2.1 Einführung

Um die Funktionalität des Process Mining Framework ProM anhand der durch den Protokollierungsmechanismus (siehe 4.3) erzeugten Prozessdaten zu demonstrieren, wurde eine ent-

sprechende Beispielanwendung entwickelt. Hierfür wurde die Standard *Apache Camel Tracer*-Beispielanwendung [9] als Ausgangsbasis gewählt. Diese Beispielanwendung definiert eine simple *Apache Route* (siehe 4.3.2), welche aufgrund des überschaubaren Umfangs in weiterer Folge zu einem übersichtlichen und verständlichen Prozessmodell führt. Um einen größeren Umfang der ProM Funktionalität zeigen zu können, wurde diese *Apache Route* der Basisanwendung um zusätzliche Funktionalität erweitert. Des Weiteren wird die beschriebene Mockup-Funktionalität zur Erzeugung von Akteur-Informationen verwendet, um die Ableitung von Prozessmodellen der Organisatorischen Perspektive zu ermöglichen.

5.2.2 Funktionaltität der Beispielanwendung

Die Beispielanwendung ermittelt aus einer beliebigen Anzahl von Wörtern (diese werden via Kommandozeile durch Leerzeichen getrennt vom Benutzer eingegeben) ein Wort, für welches ein der Applikation bekanntes Zitat an der Standardausgabeschnittstelle ausgegeben wird. Im Hintergrund ermittelt die Applikation für die restlichen Wörter ebenfalls ein passendes Zitat, diese Ergebnisse werden dem Benutzer jedoch nicht preisgegeben. Welches Zitat im Endeffekt ausgegeben wird, erfolgt rein zufällig. Sowohl die bekannten Wörter als auch die bekannten Zitate sind in der Konfigurationsdatei der Applikation definiert. Ist der Applikation ein eingegebenes Wort nicht bekannt bzw. ist für dieses kein Zitat hinterlegt, wird anstatt des Zitates ein entsprechender Hinweis für den Benutzer ausgegeben (falls das Wort zufällig ausgewählt wurde). Die Anwendung erlaubt es, mehrmals hintereinander eine Liste an Wörtern einzugeben bis der Benutzer die Anwendung selbstständig beendet.

Die entsprechende *Apache Route* der Beispielanwendung ist in 5.1 illustriert.

```
1 <route>
2   <!-- in stream to allow you to enter some text in the console -->
3   <from uri="stream:in?initialPromptDelay=4000&amp;
4 ␣␣␣promptDelay=2000&amp;␣promptMessage=Enter␣some␣words:"/>
5
6   <!-- split the text using parallel execution -->
7   <split parallelProcessing="true">
8     <!-- use methodCall expression to split the words,
9   using a java bean to do it -->
10    <method bean="quoteService" method="splitWords"/>
11
12    <!-- for each split message invoke the quote server to get
13    a quote of the word -->
14    <to uri="bean:quoteService?method=quote"/>
15
16    <!-- add the internal counter to the quote -->
17    <to uri="bean:quoteService?method=increment"/>
18
19     <!-- now we need to find the best quote,
```

```
20      so we aggregate all the splitted words
21            we use our own strategy how to aggregate -->
22        <aggregate strategyRef="myAggregateStrategy">
23            <!-- correlate everything using constant true,
24      as they are all from the same source -->
25            <correlationExpression>
26              <constant>true</constant>
27            </correlationExpression>
28            <!-- complete after 1 sec on inactivity -->
29            <completionTimeout>
30                <constant>1000</constant>
31            </completionTimeout>
32            <!-- send the result to stream out so we can see
33          the response in the console -->
34            <to uri="stream:out"/>
35        </aggregate>
36      </split>
37  </route>
```

Listing 5.1: Beispielanwendung - Apache Camel Route

Die definierte *Route* umfasst mehrere Verarbeitungsschritte, wie in der folgenden Liste beschrieben. Der Verarbeitungsschritt *quoteService.increment* (im Zuge der parallelen Ausführung) ist eine Erweiterung der ursprünglichen Implementierung der Beispielanwendung, alle restlichen sind bereits in der verwendeten Basisanwendung definiert:

- *stream:in* - Eingabe eines beliebigen Textes über die Kommandozeile
- *quoteService.splitWords* - Aufteilung der Zeichenkette in einzelne Wörter durch das *QuoteService (splitWords)*
- Parallel für jedes Wort:
  - *quoteService.quote* - Ermittlung eines passenden Zitates durch das *QuoteService (quote)*
  - *quoteService.increment* - Ergänzung des Zitates um den aktuellen internen Zähler des QuoteServices (increment)
- *aggregate* - Nach Prozessdurchführung aller Wörter wird zufällig eines der ermittelten Zitate ausgewählt
- *stream:out* - Ausgabe des augewählten Zitates

5.2.3 Akteurinformationen der Ereignisse

Da in der ursprünglichen Version der Beispielanwendung keine Resource-Informationen berücksichtigt wurde, musste durch die Nutzung der bereits besprochenen Mockup-Funktionalität

(siehe 4.3) die Anwendung um zusätzliche Informationen angereichert werden. Dies ermöglicht es in weiterer Folge ein Prozessmodell der organisatorischen Perspektive im ProM Framework aus den Prozessdaten ableiten zu können. Zur Erzeugung von Resource Informationen wurden zwei unterschiedliche Mockup Implementierungen eingesetzt, was es ermöglichte eine komplexere Organisationsstruktur zu simulieren:

- **Standard Resource-Mockup:** Die Logik wählt zufällig einen Akteur aus einer bekannten Liste aus oder entscheidet für manche Fälle, dass keine Akteur Information verwendet werden soll. Diese Mockup-Logik wird für jedes Ereignis ausgeführt.

- **Erweitertes Resource-Mockup:** Die erweiterte Logik fügt einen von zwei statisch bekannten Akteuren dem aktuellen Ereignis hinzu. Die Verteilung zwischen den Akteuren ist jedoch nicht linear, in 9 von 10 Fällen wird Akteur A ausgewählt. Diese Logik wird nur für den Verarbeitungsschritt *quoteService.increment* berücksichtigt, in diesem Fall wird die durch die Standard-Variante erzeugte Akteur-Information überschrieben. In allen anderen Verarbeitungsschritten wird die erweiterte Variante nicht durchgeführt.

5.2.4 Testdaten

Für die Demonstration der Funktionalität des ProM Frameworks war es notwendig, entsprechende Testdaten in einem ausreichend großen Umfang zu generieren. Hierfür wurde die beschriebene Beispielanwendung zur Erzeugung der Prozessdaten genutzt und folgendermaßen ausgeführt:

- **Beispielanwendung:** Die Anwendung wurde insgesamt fünf mal gestartet
  - **Eingabe von Wortlisten:** Pro Sitzung wurden jeweils zehn unterschiedliche Wortlisten eingegeben
  - **Wortlisten:** Eine Wortliste enthielt zwischen ein und vier Wörter
- **Apache Camel Route:** Folglich ergaben sich 50 Prozessinstanzen mit insgesamt 1016 Ereignissen

Die entsprechenden Protokolldaten der Prozessinstanzen standen nach der Durchführung als Dokumente (in einer Collection) in der MongoDB zur Verfügung (wie in 4.3] besprochen) und wurden in weiterer Folge mittels des beschriebenen Import- und Konvertierungsmechanismus (siehe 4.4) in das XES-Format überführt und in den Objekt-Pool des ProM Frameworks geladen. In den folgenden Absätzen wird die ProM Framework-Funktionalität zur Ableitung von Prozessmodellen in den unterschiedlichen Perspektiven anhand der erzeugten Prozessdaten der Beispielanwendung illustriert.

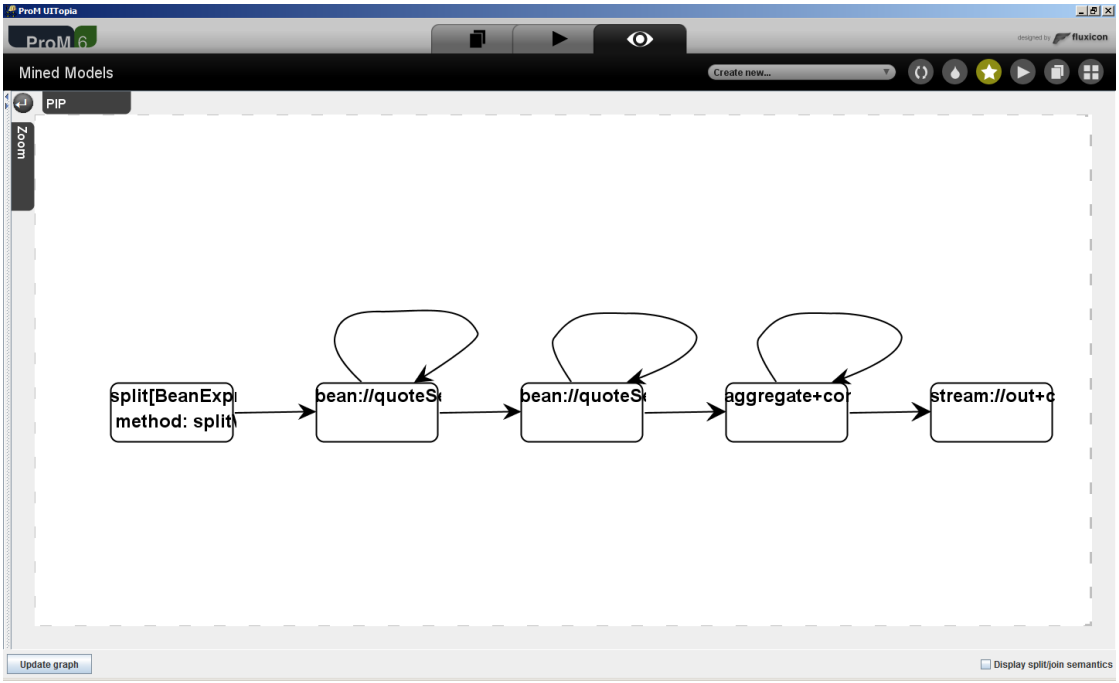

Abbildung 5.1: Flexible Heuristic Miner - Prozessmodell der Beispielanwendung

5.3 Mining der Prozess Perspekitve

Mit der Ableitung eines Prozessmodells der Prozess Perspektive kann, wie bereits schon erwähnt, der Kontrollfluss des entsprechenden Prozesses abgebildet werden. Aus den vom Protokollierungsmechanismus erzeugten Prozessdaten kann das IST-Prozessmodell extrahiert werden, welches den Prozessablauf der protokollierten Prozessdaten zeigt. Dieses Prozessmodell kann als eine Repräsentation der Mehrheit der Prozessinstanzen gesehen werden. Da in der Beispielanwendung ein relativ einfacher Prozessablauf ohne Alternativpfade betrachtet wird, ist das abgeleitete Prozessmodell folglich übersichtlich und kompakt darstellbar.

Im ProM Framework (Version 6.0) gibt es unterschiedliche Algorithmen zur Ableitung eines Prozessmodell aus der Prozess Perspektive. Beispiele für solche Algorithmen sind der α - Algorithms, der Flexible Heuristic Miner, der Genetic Miner und der Fuzzy Miner. Unterschiedliche Algorithmen, können das Prozessmodell in unterschiedlicher Art und Weise abbilden, wie beispielsweise bereits im Zuge der Diskussion zu Schleifen im Prozessmodell besprochen wurde. Für die vorhandenen Prozessdaten der Beispielanwendung wurden die unterschiedlichen Mining-Algorithmen auf den Prozesstestdaten ausgeführt. Der Fuzzy Miner sowie der Flexible Heuristic Miner 5.1 führten zu einem identischen Prozessmodell, der Genetic Miner 5.2 erzeugte im Gegensatz dazu ein Prozessmodell, welches auf den ersten Blick deutlich von den anderen Prozessmodellen abweicht. Der von der Abweichung betroffene Teil des Prozessmodells, war jener, welche die parallelen Ausführungen (Verabeitungen der einzelner Wörter) abbildete. In

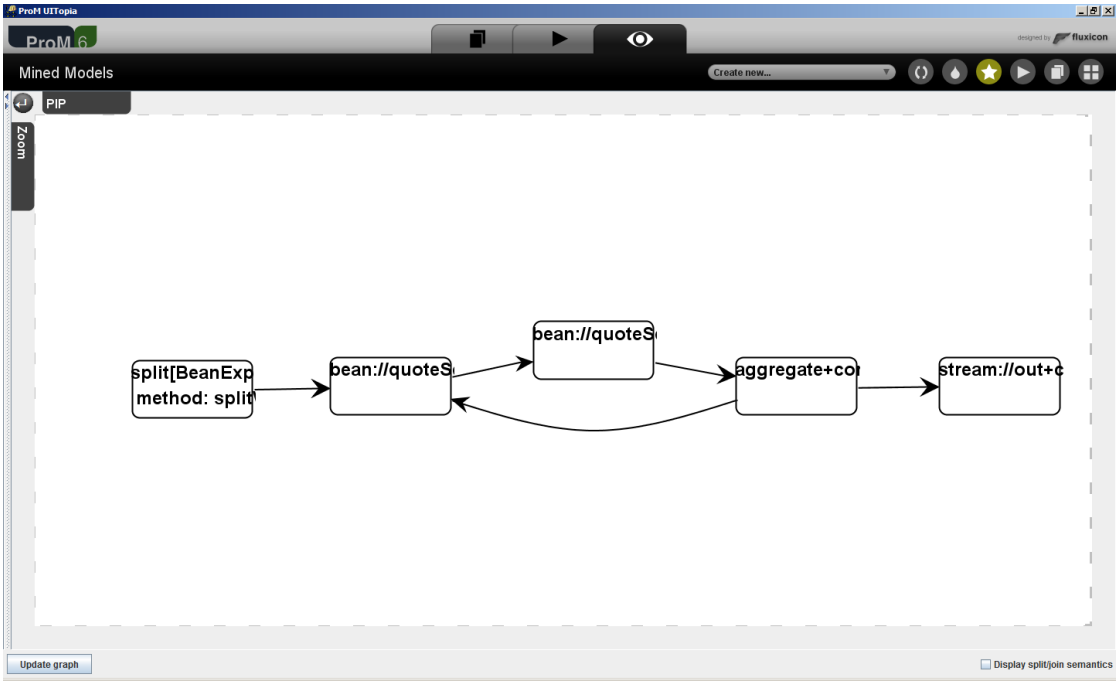

Abbildung 5.2: Genetic Miner - Prozessmodell der Beispielanwendung

allen resultierenden Prozessmodellen wurde dieser Teil des Modells als Schleife der betroffenen Ereignissen (unterschiedlich aufbereitet) abgebildet. Im Genetic Miner wird diese Schleife als Zyklus zwischen mehreren Ereignissen dargestellt, in den anderen Algorithmen werden diese als direkte Schleifen der inneren Ereignisse abgebildet.

Alle abgeleiteten Prozessmodelle zeigen wie erwartet, dass die Aufteilung der Eingabe in einzelne Wörter (*split*) das erste Ereignis im Prozessmodell, sowie die Ausgabe an der Standardausgabeschnittstelle das letzte Ereignis im Prozessablauf ist.

5.4 Mining der Organisatorischen Perspektive

Des Weiteren kann aus, durch den Protokollierungsmechanismus erzeugten, Prozessdaten (da wir die Mockups zur Erzeugung von Akteurinformationen genutzt haben) ein Prozessmodell der Organisatorischer Perspektive extrahiert werden. Die Idee der Organisatorischen Perspektive besteht darin, den Informationsfluss (z.B.: Kommunikation oder Delegation) einzelner Prozessinstanzen zwischen den Akteuren aufzuzeigen. Das ProM Framework (Version 6.0) bietet ebenfalls verschiedene Algorithmen zur Gewinnung solch eines Prozessmodells, unter anderem als Darstellung in Form eines Sozialen Netzwerks.

5.4.1 Arten von Sozialen Netzwerken

Soziale Netzwerke können aufgrund unterschiedlicher Aspekte generiert werden [7]. Für diesen Zweck werden im ProM Framework unterschiedliche Mining PlugIns bereitgestellt:

- **Handover of work (HoW):** Hierbei steht die Übergabe von Resultaten einzelner Aktivitäten für die folgenden Aktivitäten der Prozessinstanz im Vordergrund. Erledigt beispielsweise Akteur A die erste und Akteur B die zweite Aktivität innerhalb einer Prozessinstanz, spricht man von einem Handover of Work.

- **Reassignment (RA):** Dieser Aspekt zeigt die Delegation von Aktivitäten innerhalb der Prozessabläufe. Z.B.: wenn Akteur A die Durchführung der Aktivität Z immer an Akteur B delegiert, spricht man von einem Reassignment.

- **Subcontracting (SC):** Subcontracting ist der Idee von Handover of Work sehr ähnlich. Anders als bei HoW ist die Beziehung zwischen den Akteuren nicht uni- sondern bidirektional. Wird in einer Prozessinstanz zwischen zwei von Akteur A ausgeführten Aktivitäten häufig eine Aktivität von Akteur B ausgeführt, kann eine Subcontracting-Beziehung abgeleitet werden.

- **Similar task (ST):** Die Beziehung zwischen den Akteuren wird, anders als bei den bereits beschriebenen Techniken, nicht aufgrund der Ausführungssequenz der Prozessinstanzen abgeleitet, sondern das Soziale Netzwerk wird aufgrund der Gemeinsamkeit der Aktivitäten gewonnen, d.h. welche Akteure führen die gleichen Aktivitäten durch.

- **Working together (WT):** Die Beziehung der Akteure wird aus der Zusammenarbeit in den Prozessinstanzen abgeleitet. Für das resultierende Prozessmodell wird die Häufigkeit einer Zusammenarbeit betrachtet, wobei es hierfür nicht interessant ist, ob diese in einer anderen direkten Beziehung (HoW, RA, SC) stehen.

5.4.2 Kennzahlen zur Analyse

Nachdem ein Soziales Netzwerk aus den Prozessdaten abgeleitet wurde, ist es notwendig, die Kennzahlen für die Analyse dieses Netzwerks festzulegen. Die unterschiedlichen Kennzahlen ermöglichen es, die Rolle bzw. den Einfluss der einzelnen Akteure innerhalb des Netzwerks zu analysieren (z.B.: ist ein Akteur eine zentrale Stelle oder vom Netzwerk isoliert). Laut [7] können diese Kennzahlen in zwei Kategorien eingeteilt werden: 1) Kennzahlen auf Ebene der Akteure und 2) Kennzahlen auf Ebene des Netzwerks. In weiterer Folge wird nur ein Auszug von Kennzahlen der ersten Kategorie (da nur diese auch im ProM Framework vorhanden sind) berücksichtigt:

- **Degree:** Diese Kennzahl zeigt den Grad an Beziehungen zu einem Akteur, d.h die Popularität dessen im Netzwerk.

- **Betweenness:** Zeigt den Einfluss, den ein Akteur für die Informationsverteilung im Netzwerk hat. Ein Akteur mit einem hohen Grad an Betweenness spielt eine kritische Rolle

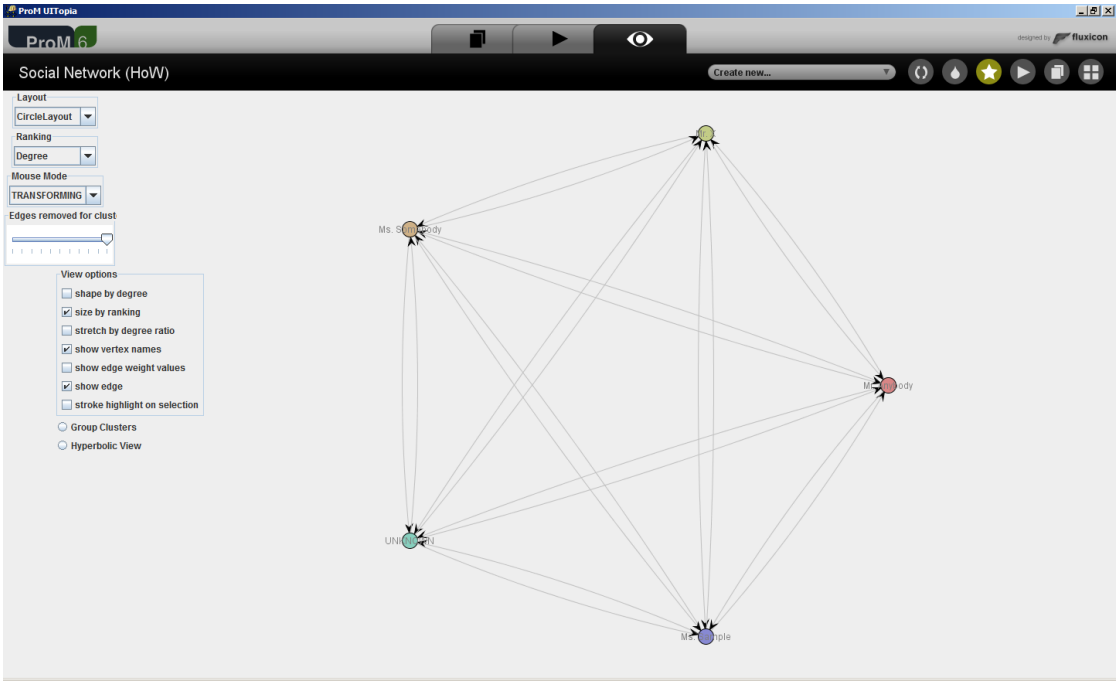

Abbildung 5.3: Social Network (HoW) Miner - Ergebnis für die Daten der Beispielsanwendung

im Netzwerk, da dieser als Verbindungsstelle zwischen zwei Gruppen agiert. Fällt dieser Akteur aus, hat dies eine direkt Auswirkung auf den Informations- und Wissensaustausch im Netzwerk.

- **Closeness:** Mit dieser Kennzahl kann die Nähe eines Akteurs zu den restlichen im Netzwerk abgebildet werden. Ist diese Kennzahl hoch, bedeutet dies, dass der jeweilige Akteur eine weniger zentrale Rolle im Netzwerk einnimmt. Für Wissens- oder Informationsbeschaffung muss dieser Akteur viele andere Akteure im Netzwerk kontaktieren. Ist die Kennzahl Closeness niedrig, nimmt der Akteur im Netzwerk eine zentrale Rolle ein.

Das ProM Framework bietet eine Visualisierungskomponente, welche es erlaubt, die Kennzahlen dynamisch zu verändern und folglich die Darstellung des Netzwerks zu aktualisieren.

5.4.3 Ergebnisse für die Beispielanwendung

Für unsere Beispielanwendung wurden zwei unterschiedliche Algorithmen zur Erzeugung eines Sozialen Netzwerks verwendet: 1) Social Network (HoW) Miner 5.3 und 2) Social Network (WT) Mine 5.4. Wie zu Beginn des Kapitels beschrieben, wurden die Atkeurinformationen durch die Verwendung von speziellen Mockups zufällig erzeugt. Die durch das Standard Mockup erzeugten unterschiedlichen Akteurinformationen, treten in den einzelnen Ereignissen der Prozessinstanzen mit einer annähernd ähnlichen Häufigkeit auf. Dies führt dazu, dass das Resultat

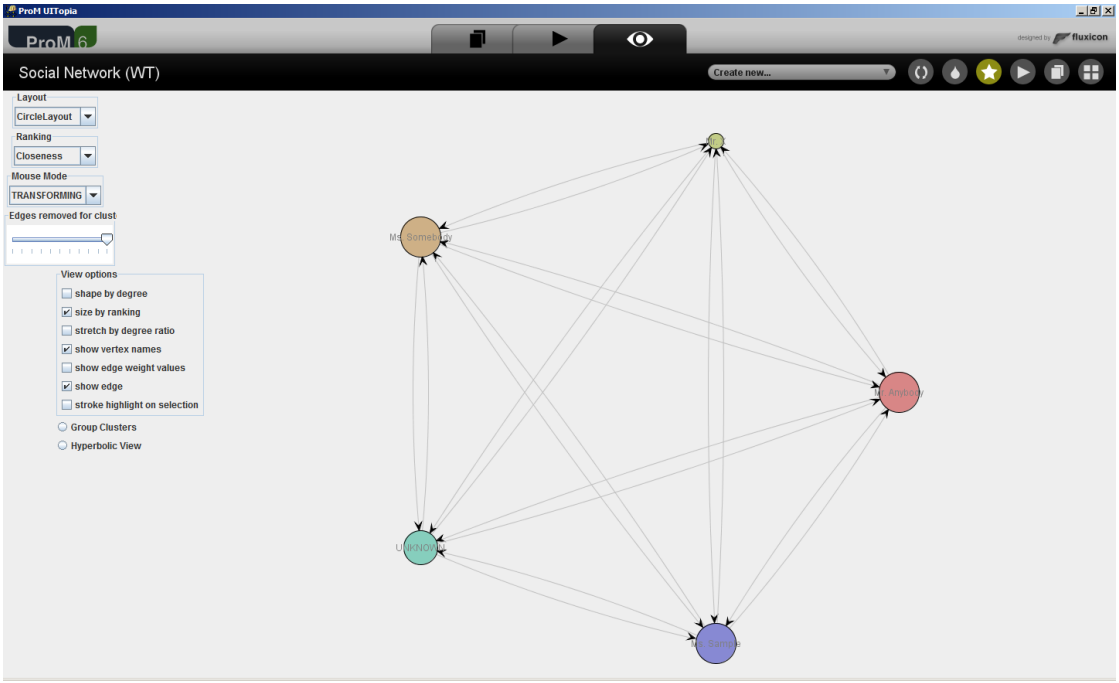

Abbildung 5.4: Social Network (WT) Miner - Ergebnis für die Daten der Beispielsanwendung

des Sozialen Netzwerks (HoW) bei der Betrachtung der Kennzahl Degree ein homogenes Netzwerk zeigt (siehe 5.3).

Im zweiten erzeugten Sozialen Netzwerk (WT) wurde die Zusammenarbeit der Akteure innerhalb von Prozessinstanzen betrachtet. Wie bereits zu Beginn dieses Kapitels beschrieben, führt ein bestimmter statisch bekannter Akteur die Aktivität *quoteService.increment* in 9 von 10 Fällen durch (erweitertes Mockup). Betrachtet man das Soziale Netzwerk (WT) nun im Kontext der Kennzahl Closeness zeigt sich, dass sich für diesen besonderen Akteur die Kennzahl besonders niedrig ausprägt und folglich dieser in dem abgeleiteten Netzwerk eine zentrale Rolle einnimmt (siehe 5.4).

KAPITEL 6

Zusammenfassung

Im Zuge dieser Diplomarbeit wurden zwei Software Artefakte a) Protokollierungsmechanismus und b) Import- und Konvertierungsmechanismus implementiert, mit welchen die Möglichkeit zur zeitnahen Nutzung von traditionellen Process Mining Anwendungsszenarien demonstriert werden konnten. Der vorgestellte Protokollierungsmechanismus *Morphia Camel Tracer* ermöglicht die Protokollierung der einzelnen Aktivitäten einer Apache Camel Applikation in die dokumentorientierte Datenbank *MongoDB*. Die einzelnen Ereignisse werden in Form von *Dokumenten* in einer entsprechenden *Collection* dauerhaft abgelegt und können somit in weiterer Folge als Basis für Analysen verwendet werden. Der Import- und Konvertierungsmechanismus *Morphia Camel Tracer ProM Import PlugIn* erlaubt es, das durch den Protokollierungsmechanismus erzeugte Ereignisprotokoll in das *XES Format* (XML Standard zu Repräsentation von Ereignisprotokollen) zu überführen und dieses konvertierte Ereignisprotokoll in weiterer Folge in den Objekt-Pool des *ProM Frameworks* zu laden. Der Importmechanismus erlaubt es dem Benutzer die Verbindungskonfiguration zur Datenquelle individuell vorzunehmen sowie Konvertierungs- und Filterparameter entsprechend der jeweiligen Anforderung zu definieren. Sobald das deklarierte Ereignisprotokoll im Objekt-Pool des *ProM Frameworks* vorliegt, kann dieses für jegliche Mining- und Analyse-Algorithmen als Eingangsobjekt verwendet werden.

Die geschaffene Lösungsarchitektur (siehe Kapitel 4) erlaubt eine zeitnahe Nutzung des Ereignisprotokolls von Apache Camel Applikationen (welche den Protokollierungsmechanismus nutzen) für die Anwendung von Process Mining Techniken im *ProM Framework*, wobei die Durchführung aller im Zuge dieser Diplomarbeit beschriebenen traditionellen Anwendungsszenarien (siehe 2) möglich ist. Im Vergleich zu bestehenden traditionellen Lösungsansätzen kann durch die gewählte Lösungsarchitektur, die Extraktion der Ereignisprotokolldaten in ein externes Datenformat automatisiert durchgeführt werden. Dies erlaubt einerseits eine zeitnahere Nutzung dieser Prozessdaten in Process Mining Techniken und reduziert andererseits den Aufwand für diesen, in den meisten Fällen manuell durchgeführten, Schritt. Folglich kann ein Protokollausschnitt der Prozessdaten, sofort nach der Erzeugung durch den Protokollierungsmechanimus, in das ProM Framework geladen und für Analysen als Basis genutzt werden. Des Weiteren wur-

de die dokumentorientierte Datenbank *MongoDB* als Datenbasis für die Ereignisprotokolldaten gewählt, was, wie bereits in Kapitel 4 diskutiert wurde, erhebliche Effizienzvorteile gegenüber anderen Möglichkeiten der Datenspeicherung (z.B.: relationale Datenbank) bietet. Die Entscheidung zur Nutzung der *MongoDB* wirkte sich somit positiv auf die Effizienz der gewählten Lösungsarchitektur aus. Die Entscheidung zur Nutzung der *MongoDB* konnte außerdem durch den Umstand, dass die idente Technologie in dem bereits erwähnten bestehenden universitären Projekt eingesetzt wird, begründet werden. Diese Technologieauswahl führt in diesem Projekt bei Einsatz des Protokollierungsmechanismus zu einer homogenen Gesamtarchitektur.

Eine zusätzliche Überlegung war es, mit Hilfe der Lösungsarchitektur den abschließenden Arbeitsschritt traditioneller Anwendungsszenarien, die Durchführung von Mining- und Analyse-Algorithmen, ebenfalls zu automatisieren. Solche eine Automatisierung dieses Schrittes hätte es erfordert die Nutzungsmöglichkeiten (d.h. welcher Algorithmus soll für die Daten ausgeführt werden) der Mining- und Analyse-Algorithmen einzuschränken, was sich negativ auf die Flexibilität der Lösung ausgewirkt hätte. Um diese Flexibilität zu erhalten, erlaubt die umgesetzte Lösung dem Benutzer, nach dem Import der Ereignisprotokolldaten (in den Objekt-Pool des ProM Framework) jeden beliebigen Algorithmus auf dieses Ereignisprotokoll anzuwenden. Somit bleibt die Mächtigkeit des ProM Frameworks, dass dieses durch die Möglichkeit des flexiblen Einsatzes unterschiedlicher Algorithmen bietet, erhalten (wie in Kapitel 5 an einem Beispiel illustriert).

Zusätzlich zu den in der Literatur bekannten Herausforderungen des Process Minings, konnte im Zuge der Umsetzung der Lösungsarchitektur eine weitere Herausforderung identifiziert werden. Im Zuge der Analysen, der durch die erste Version des Protokollierungsmechanismus erzeugten Ereignisprotokolldaten, zeigte sich, dass keine eindeutigen Verbindungen zwischen Prozessinstanzen und den zugehörigen Ereignissen vorhanden waren (siehe 2). Auf Basis dieser Prozessdaten war es folglich nicht möglich ein valides Ereignisprotoll abzuleiten. Daher war es erforderlich den Protokollierungsmechanismus um eine zusätzliche technische Lösung zu erweitern, welche den protokollierten Ereignissen die jeweilige Prozessinstanznummer immer zwingend hinzufügt (wie in Kapitel 4 vorgestellt). Eine weitere Lösungsmöglichkeit war, die Aufgabe der Verwaltung der Prozessinstanznummern der jeweiligen Apache Camel Applikation zu überlassen, was jedoch den Einsatz des Protokollierungsmechanismus ohne individuelle Anpassung der Apache Camel Anwendung nicht erlaubt hätte.

Die Lösungsarchitektur erlaubt in der aktuell umgesetzten Form, keine Unterstützung von Anwendungsszenarien zur operativen Unterstützung durch Process Mining (siehe Kapitel 3). Jedoch sind die Teile der geschaffenen Software Komponenten ohne umfassende Anpassungen auch für diesen Zweck einsetzbar. In der gewählte Lösungsarchitektur besteht eine zwingende unidirektionale Abhängigkeit zwischen den Komponenten. Der Import- und Konvertierungsmechanismus kann nur auf Basis eines durch den Protokollierungsmechanismus erzeugten Ereignisprotokolls verwendet werden (da dieser die Ereignisse als Dokumente einer spezifischen Form erwartet), der Protokollierungsmechanismus ist im Gegensatz jedoch unabhängig von der Komponente, welche die Prozessdaten konsumiert. Die umgesetzte Importlogik ist außerdem

nur im Process Mining Framework ProM einsetzbar, die Konvertierungslogik der Komponente wäre jedoch im Gegensatz zur Importlogik (diese müsste zur operativen Unterstützung automatisiert werden) auch für einen Einsatz zur operativen Unterstützung modifiziert einsetzbar. Auch der Einsatz der dokumentorientierten Datenbank *MongoDB*, würde durch die besprochene Effizienz, eine operative Unterstützung durch eine modifizierte Lösung begünstigen. Mit der Entwicklung einer entsprechenden erweiterten Importkomponente ist, der Ansatz einer operativen Unterstützung durch Process Mining, mit Komponenten (bzw. Teilen dieser) der vorgestellten Lösung grundsätzlich möglich, weiterführende Herausforderungen müssten jedoch noch vertiefend diskutiert und gelöst werden.

Ein weiteres Ergebnis dieser Diplomarbeit war ein Vorschlag eines Vorgehensmodells für die Umsetzung von Process Mining (siehe Kapitel 2). Dieses vorgeschlagene Vorgehensmodell sieht eine Strukturierung der Umsetzung in mehrere Teilphasen vor: 1) Anforderungsanalyse, 2) Datenselektion, 3) Aufbereitung der Ereignisprotokolldaten, 4) Evaluierung der Process Mining Techniken, 5) Konvertierung der Daten, 6) Implementierung und Beobachtung und 7) Visualisierung der Ergebnisse. Die gewählte Strukturierung orientiert sich an bestehenden Data Mining Prozessen, wobei für das Process Mining spezifische Charakteristiken zusätzlich berücksichtigt wurden. Im Zuge der Entwicklung der Lösugngsarchitektur wurden nach dem vorgeschlagenen Vorgehensmodell vorgegangen.

Literaturverzeichnis

[1] Bichler, M. Design Science in information systems research. *Wirtschaftsinformatik*, 48:133–135, 2006. 10.1007/s11576-006-0028-8.

[2] Casati, F., Dayal U., Sayal, M. and Shan M.C. Business Process Intelligence. 2002.

[3] Castellanos, M., Alves de Medeiros, A.K., Mendling, J., Weber, B., Weijters, A. J. M. M. *Business Process Intelligence*, chapter Chapter XXI, pages 467–491. IGI Global, 2009.

[4] Chodorow, K. and Dirolf, M. *MongoDB: The Definitive Guide*. OReilly Media, 2010.

[5] Cook, J.E. and Wolf, A. L. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Volume 7 Issue 3, 1998.

[6] Crockford, D. JavaScript Object Notation (JSON) - Standard: http://tools.ietf.org/html/rfc4627, July 2006.

[7] da Costa Alves, C.S. Social Network Analysis for Business Process Discovery. Master's thesis, Universidade Tecnica de Lisboa, 2010.

[8] DeMichiel, L. Enterprise JavaBeans 3.0 Final Release, 2007.

[9] Apache Camel Tracer Example. http://camel.apache.org/tracer-example.html, July 2011.

[10] Guenther, C.W. *XES eXtensible Event Stream Developer Guide*, 2009.

[11] Guenther, C.W. XES eXtensible Event Stream Standard Definition, 2009.

[12] Guenther, C.W. and van der Aalst, W.M.P. A Generic Import Framework for Process Event Logs. In Johann Eder and Schahram Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 81–92. Springer Berlin / Heidelberg, 2006.

[13] Guo, Q., Vaidya, J. and Atluri, V. The Role Hierarchy Mining Problem: Discovery of Optimal Role Hierarchies. *ACSAC '08 Proceedings of the 2008 Annual Computer Security Applications Conference*, 2008.

[14] Gupta, G.K. *Introduction to Data Mining with Case Studies, Chapter 1*. Prentice-Hall of India Pvt.Ltd, September 2006.

[15] Hohpe G. and Woolf B. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2003.

[16] Ibsen, C. and Anstey, J. *Camel in Action*, chapter 1, pages 4–25. Manning, 2010.

[17] ICATPN, editor. *Proceedings of Applications and Theory of Petri Nets 2004: 25th International Conference*, number Volume 3099. Springer-Verlag, 2004.

[18] Keller, G. and Teufel, T. *SAP R/3 Process Oriented Implementation*. Addison-Wesley Longman Publishing Co., 1998.

[19] Korb, P. Realization of EAI Patterns with Apache Camel. Master's thesis, Universitaet Stuttgart, 2008.

[20] Ly, L.T. Process Mining: Bestehende Ansaetze und weiterfuehrende Aspekte. Master's thesis, Universitaet Ulm, 2005.

[21] Medeiros, A. K., Weijters, A. J. and Aalst, W. M.P. Genetic Process Mining: An experimental evaluation. *Data Mining and Knowledge Discovery*, Volume 14 Issue 2, 2007.

[22] Munoz-Gama, J. Algorithms for process conformance and process refinement. Master's thesis, Universitat Politecnica de Catalunya, 2010.

[23] Oracle. Core J2EE Patterns - Data Access Object. http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html, July 2011.

[24] Reisig, W. and Rozenberg, G. Lectures on Petri Nets I: Basic Models. *Lecture Notes in Computer Science*, Volume 1491, 1998.

[25] Rozinat, A. and van der Aalst, W.M.P. Conformance checking of processes based on monitoring real behavior. *Information Systems*, Volume 33 Issue 1, 2008.

[26] IDS Scheer. ARIS Process Performance Monitor (ARIS PPM): Measure, Analyze and Optimize your Business Process Performance (whitepaper), 2002.

[27] Schonenberg, M.H., Weber, B. and van Dongen, B.F. and van der Aalst, W.M.P. Supporting Flexible Processes through Recommendations Based on History. In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, *Business Process Management*, volume 5240 of *Lecture Notes in Computer Science*, pages 51–66. Springer Berlin / Heidelberg, 2008.

[28] Song, M. and van der Aalst, W.M.P. Towards comprehensive support for organizational mining. *Decision Support Systems*, Volume 46 Issue 1, 2008.

[29] Thuraisingham, B., Khan, L., Clifton, C.T., Maurer, J. and Ceruti, M. Dependable Real-Time Data Mining. *ISORC '05 Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 2005.

[30] van der Aalst, W. M. P., Reijers, H. A. and Song, M. Discovering Social Networks from Event Logs. *Computer Supported Cooperative Work (CSCW)*, Volume 14:pages 549–593, 2005.

[31] van der Aalst, W. M. P., Weijters, A. J., and Maruster L. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, Volume 16 Issue 9, 2004.

[32] van der Aalst, W.M.P. and Song, M. Mining Social Networks: Uncovering Interaction Patterns in Business Processes. In Jaeg Desel, Barbara Pernici, and Mathias Weske, editors, *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer Berlin / Heidelberg, 2004.

[33] van der Aalst, W.M.P. and Weijters, A.J. Process Mining: A Research Agenda. *Computers in Industry - Special issue: Process/workflow mining*, Volume 53 Issue 3:231 – 244, 2004.

[34] van der Aalst, W.M.P., Pesic, M. and Song, M. Beyond Process Mining: From the Past to Present and Future. In B. Pernici, editor, *Advanced Information Systems Engineering*, volume 6051 of *Lecture Notes in Computer Science*, pages 38–52. Springer Berlin / Heidelberg, 2010.

[35] van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., Alves de Medeiros, A.K., Song, M. and Verbeek, H.M.W. Business process mining: An industrial application. *Information Systems*, Volume 32 Issue 5:713–732, 2007.

[36] van der Aalst, W.M.P., Schonenberg, M.H. and Song, M. Time Prediction Based on Process Mining. *Information Systems*, Volume 36 Issue 2, 2011.

[37] van der Werf, J., van Dongen, B.F., Hurkens, C. and Serebrenik, A. Process Discovery Using Integer Linear Programming. In K. van Hee and R. Valk, editors, *Applications and Theory of Petri Nets*, volume 5062 of *Lecture Notes in Computer Science*, pages 368–387. Springer Berlin / Heidelberg, 2008.

[38] van Dongen, B. F. and van der Aalst, W. M. P. A Meta Model for Process Mining Data. In -, volume Volume 161, 2005.

[39] van Dongen, B.F., Crooy, R. and van der Aalst, W.M.P. Cycle Time Prediction: When Will This Case Finally Be Finished? In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems: OTM 2008*, volume 5331 of *Lecture Notes in Computer Science*, pages 319–336. Springer Berlin / Heidelberg, 2008.

[40] Verbeek, H.M.W. and Jagadeesh Chandra Bose, R.P. *ProM 6 Tutorial*, 2010.

[41] Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F. and van der Aalst, W.M.P. *XES, XESame, and ProM 6*, 2010.

[42] Apache Camel Project Website. http://camel.apache.org/, April 2011.

[43] Apache Maven Website. http://maven.apache.org/, July 2011.

[44] BSON Website. http://bsonspec.org/, July 2011.

[45] ERPEL Website. http://erpel.at//, July 2011.

[46] Hibernate Project Website. http://www.hibernate.org, July 2011.

[47] MongoDB Project Website. http://www.mongodb.org/, April 2011.

[48] Morphia Project Website. http://code.google.com/p/morphia/, April 2011.

[49] OpenXES Project Website. http://code.deckfour.org/xes/, July 2011.

[50] Process Mining Website. http://www.processmining.org/, May 2011.

[51] ProM Developer Guide Website. http://www.processmining.org/prom/developers, July 2011.

[52] Spring Project Website. http://www.springsource.org/, July 2011.

[53] Weijters, A. J., Aalst, W. M.P and Medeiros, A. K. Process Mining with the heuristicsminer algorithm. *BETA Working Paper Series WP 166*, 2006.

[54] Weijters, A.J. and van der Aalst, W.M.P. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering*, Volume 10 Issue 2:151 – 162, 2003.