JOHANNES KEPLER
UNIVERSITÄT LINZ

Eingereicht von
Manuel Hochreiter, BSc

Angefertigt am
Institut für Wirtschaftsinformatik Data & Knowledge Engineering

Betreuer
o. Univ.-Prof. DI Dr. Michael Schrefl

Mitbetreuung
Dr. Christoph Schütz

November 2016

Umsetzung ausgewählter Methoden für die quantitative und qualitative Analyse von mehrstufigen Geschäftsprozessen mittels XQuery

Ist- und What-If-Analyse der Durchlaufzeit sowie Fehler-Ursachen-Analyse

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science

im Masterstudium

Wirtschaftsinformatik

JOHANNES KEPLER UNIVERSITÄT LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, 21. November 2016

Manuel Hochreiter

Kurzfassung

Um die Geschäftsprozesse eines Unternehmens, die häufig auf hierarchisch geordnete Abstraktionsebenen verteilt sind, darstellen zu können sind herkömmliche Prozessmodelle oftmals nicht ausreichend. Ein *Multilevel Business Artifact* (MBA) vereint Daten- und Prozessmodelle verschiedener Hierarchieebenen in einem Objekt. Die Prozesse sind dabei als Zustandsautomaten nach der W3C-Empfehlung *State Chart XML* (SCXML) definiert, wodurch diese semi-automatisch von einer Process Engine ausgeführt werden können. Die vorliegende Arbeit beschreibt die Umsetzung eines XQuery-Moduls zur quantitativen und qualitativen Analyse von mittels MBAs modellierten mehrstufigen Geschäftsprozessen auf Grundlage der bei semi-automatischen Prozessausführungen anfallenden Logs. Zu diesem Zweck rekonstruiert das Modul die durchlaufenen MBA-Lebenszyklen unter Verwendung der Logs. Das Modul ermöglicht zum einen die Berechnung der durchschnittlichen Durchlaufzeit von Prozessen bzw. einzelnen Zuständen und zum anderen bietet es Funktionen an, um Schwachstellen in Prozessen zu identifizieren.

Abstract

Business processes are usually spread over various levels of abstraction. This leads to the fact that often they cannot be represented adequately by using conventional process models. A *Multilevel Business Artifact* (MBA) combines both data and process models of different hierarchical levels in one object. The processes are defined as state charts using the W3C recommendation of *State Chart XML* (SCXML), which allows them to be executed semi-automatically by a process engine. This thesis describes the implementation of an XQuery module for analyzing multilevel business processes which have been modelled using MBAs, based on the logs produced during semi-automatic execution of the processes. Therefore, it reconstructs the lifecycles of MBAs using logs. On the one hand, the module calculates average cycle times of processes and states and on the other hand, it provides functions to identify weaknesses of processes.

Inhaltsverzeichnis

1 Einleitung **1**

2 Hintergrund **3**
2.1 Mehrstufige Prozessmodelle 3
2.1.1 Konkretisierungshierarchien 5
2.1.2 Synchronisation von mehrstufigen Prozessmodellen 8
2.2 XML-Repräsentation mehrstufiger Geschäftsprozesse 11
2.2.1 XML, XPath und XQuery 11
2.2.2 Multilevel Business Artifacts 14
2.2.3 Prozessmodelle 16
2.2.4 Prädikate für Synchronisationsabhängigkeiten 19
2.2.5 Logging 21
2.3 Prozessanalyse 24
2.3.1 Quantitative Analyse 24
2.3.2 Qualitative Analyse 30
2.3.3 Analyse von Multilevel Business Artifacts 35
2.3.4 Schleifenerkennung 39

3 Quantitative Analyse **42**
3.1 Verwendung 42
3.1.1 Ist-Analyse 42
3.1.2 What-If-Analyse 44
3.2 Technische Dokumentation 46
3.2.1 Durchschnittliche Durchlaufzeit von Prozessen 46
3.2.2 Durchschnittliche Durchlaufzeit zusammengesetzter Zustände 56
3.2.3 Bestimmung eingehender Transitionen 59
3.2.4 Implementierung der Schleifenerkennung 62
3.2.5 Transitionswahrscheinlichkeiten 66

4 Qualitative Analyse **72**
4.1 Verwendung 72
4.2 Technische Dokumentation 73
4.2.1 Bestimmen von problematischen Zuständen 74
4.2.2 Umsetzung der Fehler-Ursachen-Analyse 74

5 Fazit und Ausblick **83**

6 Literatur **84**

Abbildungsverzeichnis

1 Konkretisierung 4
2 Spezialisierung von Transitionen mit Zielzustand 7
3 Spezialisierung von Transitionen ohne Zielzustand 8
4 Synchronisationsbeziehungen Vorbedingungen (nach [Schuetz, 2015, S. 85 ff.]) 9
5 Beispiel für Modellierung von Wartezuständen 25
6 Sequentielles Prozessmodell 26
7 Prozessmodell mit alternativen Pfaden 26
8 Prozessmodell mit einer Schleife 27
9 Prozessmodell mit `parallel`-Element 27
10 Vergleich relative und absolute Transitionswahrscheinlichkeiten . . . 28
11 Absolute Transitionswahrscheinlichkeiten bei Prozess mit Schleife . 29
12 Baumdiagramm der Fehler-Ursache-Analyse (nach [Dumas et al., 2013, S. 196 f.]) 31
13 Ursachen von Problemzustand: Subzustand 32
14 Ursachen von Problemzustand: Synchronisationsabhängigkeiten . . 34
15 Durchlaufzeiten einer hetero-homogenen Konkretisierungshierarchie 38
16 Schleifen 39
17 Index und *Lowlinks* von SZKs 40
18 Verschachtelte Schleife 41
19 Beispielprozess eines MBAs 51
20 Prozessmodell mit Schleife 54
21 Initialzustand eines `state`-Elements 61
22 Transitionen zu einem `parallel`-Element 62
23 Verlassen von Zuständen durch Transitionen 67

Tabellenverzeichnis

1 Prädikate für vertikale Synchronisationsbeziehungen über Zustände [Schuetz, 2015, S. 83 f.] 10
2 XQuery-Funktionen für vertikale Synchronisationsbeziehungen über Zustände (nach [Schuetz, 2015, S. 150 ff.]) 21
3 Logging-Elemente (nach [Schuetz, 2015, S. 188 ff.]) 23
4 Schnittstelle quantitative Analyse 43
5 Schnittstelle qualitative Analyse 73

Quellcodeverzeichnis

1 Verfeinern von Transitionsbedingungen 7
2 XML-Dokument 12
3 XPath-Ausdruck 12
4 FLWOR-Ausdruck 12
5 Signatur der Built-In Funktion `fold-left` 14
6 XML-Darstellung einer Konkretisierungshierarchie 15
7 XML-Darstellung des Daten- und Prozessmodells eines MBAs . . . 17
8 Zusammengesetzter Zustand 19
9 Parallele Zustände 19
10 Transition mit Vorbedingung 20
11 Event-Log 22
12 State-Log 23
13 State-Logs des MBAs aus Abbildung 14 34
14 State-Logs von Konkretisierungen des MBAs von Abbildung 14 der Ebene „l2“ 34
15 Kennzeichnung von *Archival States* 45
16 Format des Parameters `$changedStates` 45
17 Funktion getTotalActualCycleTime 47
18 Funktion getCycleTimeOfInstance 48
19 Funktion getAverageCycleTime 49
20 Funktion getTotalCycleTimeInStates 50
21 Funktion getTotalCycleTime 51
22 Funktion getTotalCycleTimeToState 53
23 Rückgabewert der Funktion getStateList 55
24 Funktion getStateList 55
25 Funktion getCycleTimeForCompositeState 58
26 Funktion getTransitionsToState mit zwei Parametern 59
27 Funktion getTransitionsToState mit vier Parametern 60
28 Funktion tarjanAlgorithm 63
29 Funktion strongconnect 63
30 Funktion getTransitionProbability 68
31 Funktion getTransitionProbabilityForTargetState 71
32 Rückgabewert der Funktion getCausesOfProblematicStates 73
33 Funktion getProblematicStates 75
34 Funktion getCausesOfProblematicState 76
35 Funktion getProblematicSyncs 78
36 Funktion isSyncCausingProblem 80
37 Funktion getCauseOfProblematicSync 82

1 Einleitung

In Zeiten schnell fortschreitender technologischer Entwicklung stehen Unternehmen vor der Herausforderung, sich laufend ihrem dynamischen Umfeld anzupassen. Eine Möglichkeit dazu ist die prozessorientierte Unternehmensgestaltung [Becker et al., 2009, S. 1 ff.]. Die Dokumentation von Geschäftsprozessen erlaubt deren systematische Auswertung, wodurch wertvolle Informationen über die Abläufe in Unternehmen gewonnen werden können (siehe auch [Becker et al., 2009, S. 3] und [Gadatsch, 2010, S. 11 ff.]). Die ermittelten Maßzahlen geben Auskunft über die Leistungsfähigkeit der untersuchten Prozesse und begünstigen dadurch die Entscheidungsfindung. Des Weiteren ermöglichen sie die Identifikation von Schwachstellen in Prozessen, die durch Prozessoptimierung beseitigt werden können [Dumas et al., 2013, S. 190 ff.].

Häufig sind Unternehmen hierarchisch organisiert, oftmals wird dabei eine Unterteilung in drei Ebenen verwendet (siehe dazu u. a. [Mintzberg, 1979, S. 18 ff.]). Ganz oben legt das Topmanagement die langfristigen Ziele des Unternehmens auf der strategischen Ebene fest. Darunter legt das mittlere Management mittelfristige Maßnahmen fest, um diese zu erreichen, und die operative Ebene ist für die Bewältigung des Tagesgeschäfts zuständig. Ein mehrstufiges Prozessmodell beschreibt die Ebenen eines Unternehmens mitsamt ihren Geschäftsprozessen [Schuetz, 2015]. Das Konzept des *Multilevel Business Artifact* (MBA) ermöglicht die Repräsentation von mehrstufigen Prozessmodellen einschließlich ihrer benötigten Datenstrukturen [Schuetz, 2015, S. 57 ff.] [Schütz et al., 2013]. Für die Automatisierung mehrstufiger Geschäftsprozesse existiert eine XML-basierte Notation unter Verwendung von *State Chart XML* (SCXML) [Schuetz, 2015, S. 131 ff.].

Die vorliegende Arbeit baut auf dem Konzept der MBAs auf und beschreibt die Umsetzung eines Analysemoduls für die quantitative und qualitative Auswertung von mehrstufigen Geschäftsprozessen. Das Analysemodul wurde in XQuery [W3C, 2014b] implementiert, einer Abfragesprache für XML-Dokumente/-Datenbanken. Die quantitative Analyse von Prozessen befasst sich mit der Berechnung von Maßzahlen. Das Analysemodul stellt Funktionen bereit, mit denen die durchschnittliche Durchlaufzeit von Prozessen sowie einzelnen Abschnitten berechnet werden kann. Dazu wird die Technik der *Flow Analysis* [Dumas et al., 2013, S. 213 ff.] verwendet. Diese erlaubt die Berechnung der gesamten Durchlaufzeit eines Prozesses, wenn die Durchlaufzeiten der einzelnen Aktivitäten bekannt sind. Die Ergebnisse der quantitativen Analyse werden zur Durchführung der qualitativen Analyse verwendet. Unter der Annahme, dass jeder Prozess weiter verbessert werden kann, ermittelt das Analysemodul mithilfe der Fehler-Ursachen-Analyse [Dumas et al., 2013, S. 190 ff.] Schwachstellen in den Prozessen von MBAs. Dazu werden Prozessabschnitte mit ungewöhnlich hohen Durchlaufzeiten auf ihre Ursachen untersucht.

Eine beispielhafte Ursache für eine erhöhte Durchlaufzeit sind Abhängigkeiten im Prozess, wodurch Wartezeiten entstehen können.

Abschnitt 2 erläutert die bei der Implementierung des Analysemoduls verwendeten Technologien, Methoden und Konzepte. Der erste Teil geht auf mehrstufige Prozessmodelle ein, darunter fallen Grundlagen zu mehrstufigen Prozessmodellen (2.1 Mehrstufige Prozessmodelle), ein kurzer Überblick über XPath/XQuery (2.2.1 XML, XPath und XQuery, die logische Darstellung von MBAs (2.2 XML-Repräsentation mehrstufiger Geschäftsprozesse) sowie die Aufzeichnung ihrer Lebenszyklen (2.2.5 Logging). Der zweite Teil beschäftigt sich mit Techniken zur Prozessanalyse, d.h. der quantitativen (Abschnitt 2.3.1) und qualitativen (Abschnitt 2.3.2) Analyse zuzüglich einem Abschnitt zur Analyse von MBAs (2.3.3 Analyse von Multilevel Business Artifacts). Er schließt mit der Beschreibung des Algorithmus zur Identifikation von Schleifen in Prozessmodellen (2.3.4 Schleifenerkennung). Abschnitt 3 beschreibt die Umsetzung der XQuery-Funktionen zur quantitativen Analyse von MBAs, Abschnitt 4 die der qualitativen Analyse. Abschließend gibt die Arbeit sowohl eine kurze Zusammenfassung als auch einen Ausblick auf zukünftige Arbeiten (5 Fazit und Ausblick).

2 Hintergrund

Dieses Kapitel beschreibt die bei der Umsetzung des Analysemoduls verwendeten Konzepte. Zu Beginn wird auf mehrstufige Prozessmodelle eingegangen, darauf folgt die logische Darstellung von MBAs in XML. Anschließend wird das Logging erläutert, d.h. wie die Lebenszyklen von MBAs für die nachfolgende Auswertung aufgezeichnet werden. Abschließend werden die Techniken, die bei der Analyse der Prozessmodelle zum Einsatz kommen, behandelt.

2.1 Mehrstufige Prozessmodelle

Multilevel Business Artifacts (MBAs) ermöglichen die Darstellung von Prozessen auf verschiedenen Ebenen [Schuetz et al., 2015]. Dazu vereinen sie in einem Objekt sowohl Daten- als auch Prozessmodelle, verteilt auf hierarchisch geordnete Abstraktionsebenen. Das Prozessmodell einer Ebene definiert den Lebenszyklus einer entsprechenden Instanz und legt die erlaubte Reihenfolge von Methodenaufrufen bzw. die daraus resultierenden Zustandsübergänge fest. Ein MBA beinhaltet die Daten der Instanz in der obersten Ebene (Top-Level) und beschreibt ein gemeinsames Schema für die nachfolgenden Ebenen. Diese können von anderen MBAs instanziiert werden, die dieses Schema nach den in Abschnitt 2.1.1 beschriebenen Regeln umsetzen müssen. Eine solche Instanz wird Konkretisierung eines MBAs genannt. Dadurch ist ein MBA zur gleichen Zeit ein Objekt sowie auch eine Klasse: der Top-Level beinhaltet die Daten der Instanz und die nachfolgenden Ebenen legen die Struktur weiterer MBAs fest.

Abbildung 1 zeigt eine Konkretisierungshierarchie, bestehend aus einem MBA sowie einer Konkretisierung. Sie beinhaltet das MBA „MyInsuranceCompany“ welches von „MyCarInsurance“ konkretisiert wird. MBAs sind mit mehreren Rechtecken dargestellt, die den Ebenen entsprechen und durch gestrichelten Linien verbunden sind. Die Reihenfolge entspricht dabei der Hierarchie der Ebenen im MBA. Die Bezeichnungen der Ebenen sind jeweils in spitzen Klammern vermerkt, im Top-Level wird zusätzlich die Bezeichnung des MBAs davor geschrieben. Das MBA „MyInsuranceCompany“ besitzt drei Ebenen, wobei „strategicInsurance“ dem Top-Level entspricht und damit die Instanzdaten enthält. Die übrigen Ebenen „tacticalInsurance“ und „operationalInsurance“ sind dem Top-Level untergeordnet. Die Prozessmodelle auf den Ebenen sind als UML-Zustandsautomaten modelliert [OMG, 2011]. Ein schwarzer, ausgefüllter Kreis symbolisiert einen Initialzustand. Zustände sind Rechtecke mit abgerundeten Ecken, in denen die Bezeichnung des Zustands geschrieben steht. Auf die Darstellung von finalen Zuständen wurde verzichtet, stattdessen können *Archival States* [Schuetz, 2015, S. 62] verwendet werden. Diese verhindern, dass ein MBA am Ende seiner Lebensdauer gelöscht wird. Zustandsübergänge bzw. Transitionen sind als mit dem auslösenden Ereig-

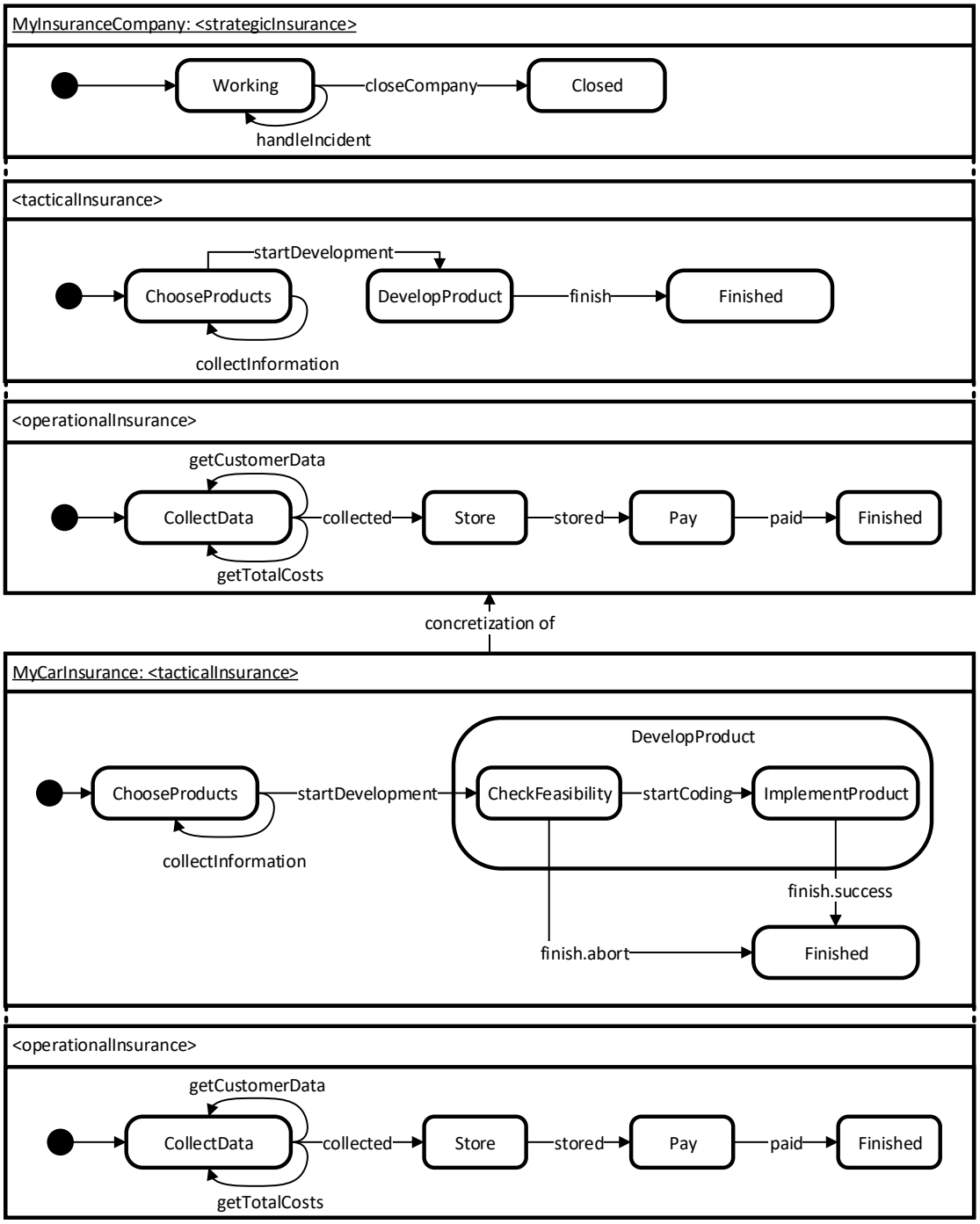

Abbildung 1: Konkretisierung

nis beschriftete Pfeile dargestellt. Eine Transition muss allerdings nicht zwingend dazu führen, dass das MBA von einem in einen anderen Zustand wechselt, z.B. verbleibt „MyInsuranceCompany“ bei dem Ereignis „handleIncident“ unverändert im Zustand „Working“. Das MBA „MyCarInsurance“ instanziiert mit „tacticalInsurance“ die zweite Ebene von „MyInsuranceCompany“. Zusätzlich verfeinert es den Zustand „DevelopProduct“, indem die beiden Subzustände „CheckFeasibility“ und „ImplementProduct“ eingeführt werden. Abschließend ist anzumerken, dass die Abbildung lediglich die Prozessmodelle der MBAs beinhaltet und auf die Darstellung der Datenmodelle verzichtet wurde. Darüber hinaus sei angemerkt, dass die konkrete Anzahl und Benennung der Ebenen je nach Anwendungsfall variiert. Die Einteilung in strategische, taktische und operative Ebene stellt dabei eine verallgemeinerte Hierarchisierung von Unternehmen dar, welche in der Praxis in unterschiedlichen Ausprägungen existiert.

2.1.1 Konkretisierungshierarchien

Dieser Abschnitt beschreibt, wie MBAs konkretisiert werden können. Des Weiteren wird erläutert, wie Prozessmodelle spezialisiert und erweitert werden können, wodurch hetero-homogene Strukturen [Neumayr et al., 2010] entstehen.

Konkretisierungen müssen den nachstehenden Regeln entsprechen [Schuetz, 2015, S. 21 ff.]. Erstens muss der Top-Level der Konkretisierung die zweite Ebene des übergeordneten MBAs sein. Das bedeutet zum einen, dass eine Konkretisierung niemals den Top-Level eines anderen MBAs instanziieren kann und zum anderen, dass in einer Konkretisierungshierarchie keine Ebene übersprungen werden kann. In einfachen Konkretisierungshierarchien muss eine Konkretisierung genau ein übergeordnetes MBA besitzen, d.h. sie kann immer nur ein MBA konkretisieren. Es ist allerdings zulässig, dass ein MBA von mehreren MBAs konkretisiert wird. Dadurch aggregiert es die Daten aller MBAs, die sowohl direkt als auch indirekt Konkretisierungen von ihm sind. In parallelen Hierarchien wäre es möglich, dass ein MBA eine Konkretisierung von verschiedenen MBAs ist, das vorliegende Analysemodul wurde jedoch für die Analyse von MBAs mit einfachen Konkretisierungshierarchien entwickelt und unterstützt keine parallelen Hierarchien.

Zweitens erben Konkretisierungen immer sämtliche Ebenen ab der zweiten Ebene des MBAs. Und drittens müssen die Prozessmodelle der Konkretisierung denen des übergeordneten MBAs entsprechen, wobei sie nach den Prinzipien der *Behaviour Consistency* spezialisiert werden dürfen [Schuetz, 2015, S. 65 f.]. Dazu kann das Prozessmodell verfeinert und erweitert werden. Bei der Verfeinerung wird der Detaillierungsgrad von bestehenden Abschnitten des Prozesses erhöht, ein Beispiel hierfür ist das Hinzufügen von Subzuständen zu einem Zustand. Im Gegensatz dazu wird dem Prozessmodell bei einer Erweiterung etwas Neues hinzugefügt, wie z.B. ein paralleler Pfad. [Schrefl und Stumptner, 2002, S. 94]

Das Prinzip der *Behaviour Consistency* wird in *Observation* und *Invocation Consistency* unterteilt. *Observation Consistency* befasst sich damit, wie ein Prozess durch Beobachtung wahrgenommen wird. Wenn alle Erweiterungen und Verfeinerungen einer Spezialisierung weggelassen werden und jeder Verlauf ihres Lebenszyklus als gültiger Lebenszyklus vom abstrakteren Modell wahrgenommen wird, liegt eine Spezialisierung nach der *Observation Consistency* vor. Angenommen ein Zustand „Bezahlung“ wird durch Hinzufügen der Subzustände „Bar“ und „Karte“ verfeinert. Das entspricht genau dann der *Observation Consistency*, wenn eine Ausführung des spezialisierten Modells, bei dem die neuen Subzustände „Bar“ und „Karte“ weggelassen werden, als korrekte Ausführung des abstrakten Modells beobachtet werden kann. *Invocation Consistency* definiert sich durch die Art, wie Spezialisierungen verwendet werden können. Die Erfüllung der schwachen *Invocation Consistency* setzt voraus, dass jede Spezialisierung gleich wie das abstraktere Modell verwendet werden kann. Die Steigerungsform davon ist die starke *Invocation Consistency*, die bedingt, dass eine Spezialisierung auch nach dem Ausführen von neuen Funktionen gleich wie das abstrakte Modell verwendet werden kann. Ein Beispiel hierfür wäre das Hinzufügen eines WLAN-Moduls zu einem Drucker, dessen Verwendung zuvor lediglich über eine Kabelverbindung möglich war. In diesem Fall ist die schwache *Invocation Consistency* erfüllt, wenn es der Drucker weiterhin erlaubt, Dokumente über Kabel auszudrucken, d.h. gleich wie zuvor verwendet werden kann. Starke *Invocation Consistency* verlangt, dass diese Funktionalität auch nach dem Verbinden des Druckers mit einem Netzwerk über das WLAN-Modul gegeben ist. [Schrefl und Stumptner, 2000, S. 528]

Das umgesetzte Analysemodul ermöglicht die Verarbeitung von MBAs sowie Konkretisierungen, die bestimmten Grundsätzen der *Observation Consistency* entsprechen, wodurch im Zuge von Konkretisierungen nur die folgenden Aktionen erlaubt sind [Schuetz, 2015, S. 105 ff.]. Erstens dürfen Zustände des übergeordneten MBAs verfeinert werden, indem Subzustände eingeführt werden. Das Weglassen von bestehenden Zuständen ist nicht zulässig. Zu den vererbten Zuständen können neue, parallele Zustände hinzugefügt werden. Dabei dürfen dem Modell allerdings keine Zustandsübergänge zwischen bereits bestehenden Zuständen hinzugefügt werden. Des Weiteren kann eine Konkretisierung neue Abstraktionsebenen erstellen, die entweder unter der letzten Ebene oder zwischen den bestehenden Ebenen eingefügt werde kann. Dies unterliegt lediglich der Einschränkung, dass die neu erstellte Ebene nicht der neue Top-Level sein darf. Einem MBA ist es erlaubt, mehrere Klassen/Prozessmodelle pro Ebene zu besitzen. Das Analysemodul ermöglicht allerdings ausschließlich die Auswertung von MBAs mit einem Klassen/Prozessmodell pro Ebene.

Die Spezialisierung von Zustandsübergängen ist ebenfalls erlaubt. Transitionen sind durch das auslösende Ereignis, die Transitionsbedingung sowie ihren

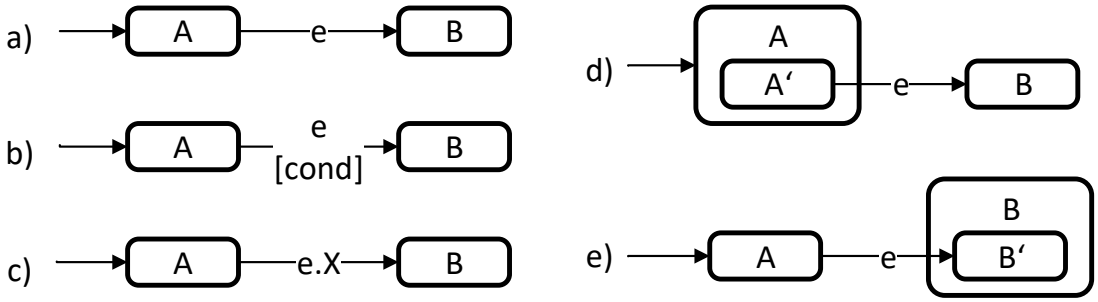

Abbildung 2: Spezialisierung von Transitionen mit Zielzustand

Ausgangs- und Zielzustand eindeutig identifiziert. Diese Merkmale müssen überprüft werden um festzustellen, ob eine Transition eine Spezialisierung einer anderen ist. Die Transition a) in Abbildung 2 zeigt einen allgemeinen Zustandsübergang zwischen den Zuständen „A“ und „B“, die durch das Ereignis „e“ ausgelöst wird. Die Transitionen von b) bis e) stellen graphisch dar, in welcher Art und Weise diese Transition a) spezialisiert werden darf. So darf ein Zustandsübergang, welcher über keine Bedingung verfügt, durch Einführung einer solchen (`cond`-Attribut, vgl. 2.2.3 Prozessmodelle) spezialisiert werden, abgebildet in b). Sollte die Transition bereits über eine Bedingung eingeschränkt sein ist es zulässig, diese durch Hinzufügen von weiteren Termen mithilfe des Schlüsselworts `and` zu präzisieren. Quellcode 1 zeigt wie die Bedingung (`cond`) einer Transition verfeinert werden kann. Die originale Transition besitzt die Bedingung „value > 3“. Die zweite, spezialisierte Transition verstärkt diese Bedingung durch Hinzufügen der zusätzlichen Klausel „value2 < 5“ mithilfe einer logischen UND-Verknüpfung (`and`). Des Weiteren kann das Ereignis, das die Transition auslöst, mithilfe von sogenannten Deskriptoren genauer definiert werden, zu sehen in Beispiel c). Diese sind mithilfe von Punkten („`.`“) getrennte Bezeichnungen von Ereignissen. Eine Transition mit dem Ereignis „event“ wird beispielsweise ebenfalls von „event.success“ ausgelöst, nicht aber durch „e.success“ oder „success“ [Barnett et al., 2015, #events]. In Punkt d) aus Abbildung 2 wird gezeigt, dass der Ausgangszustand der Transition verfeinert werden darf. Ging die Transition mit dem Ereignis „e“ in der allgemeinen Variante noch vom Zustand „A“ aus, ist der Ausgangszustand der speziellen Variante mit „A’“ nun ein Subzustand davon. Das Gleiche gilt für den Zielzustand: die Transition in Punkt e) hat „B’“ als Zielzustand, einen Subzustand des Ziels der allgemeinen Transition.

Quellcode 1: Verfeinern von Transitionsbedingungen

```
1 cond="value > 3"
2 cond="value > 3 and value2 < 5"
```

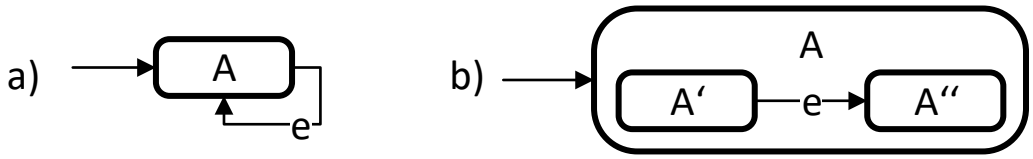

Abbildung 3: Spezialisierung von Transitionen ohne Zielzustand

Abbildung 3 illustriert den Spezialfall von Transitionen ohne Zielzustand. Bei der in Punkt a) dargestellten Transition ist der Zielzustand mit dem Ausgangszustand identisch, d.h. sie führt im Prozess zu keiner Zustandsänderung. Falls dieser Zustand verfeinert wird, ist es gemäß den zuvor beschriebenen Regeln zulässig, den Ausgangs- sowie den Zielzustand des Zustandsübergangs zu präzisieren. Demnach ist die Transition b) eine erlaubte Spezialisierung von a). Der Ausgangszustand „A'“ ist ein Subzustand der originalen Quelle „A“ und der neue Zielzustand „A”“ ist ein Subzustand des alten Ziels, welches ebenfalls „A“ war.

Damit Transitionswahrscheinlichkeiten (siehe auch 2.3.1 Quantitative Analyse) vom Analysemodul korrekt berechnet werden können, muss die Spezialisierung von Transitionen immer vollständig erfolgen. Das bedeutet, dass wenn eine Konkretisierung einen Zustandsübergang verfeinert, diese die Transition in der ursprünglichen Form nicht mehr enthalten darf.

Ein MBA beschreibt ein gemeinsames, für alle Konkretisierungen gültiges Prozessmodell, welches weiter spezialisiert werden kann. Dadurch können Variationen oder Sonderfälle eines Prozesses modelliert und der Detaillierungsgrad erhöht werden. Die daraus resultierende Hierarchie wird als hetero-homogen bezeichnet [Neumayr et al., 2010]. Die Hierarchie ist homogen in Bezug auf die Modelle, die im obersten MBA definiert wurden und dadurch Gültigkeit in sämtlichen Konkretisierungen besitzen. Durch Spezialisierungen entstehen von diesem abweichende, heterogene Modelle, die für die Subhierarchien der Konkretisierungen wiederum homogene Modelle darstellen. Konkretisierungshierarchien können beliebig weit fortgesetzt werden, da nach dem hetero-homogenen Ansatz jede Konkretisierung im Zuge einer Spezialisierung neue Ebenen einführen kann [Schuetz, 2015, S. 21 ff.]. In einer homogenen Hierarchie würde dies dem abstraktesten MBA vorbehalten sein.

2.1.2 Synchronisation von mehrstufigen Prozessmodellen

Es ist möglich, dass die Prozessmodelle der verschiedenen Ebenen eines MBAs voneinander abhängig sind. Diese vertikalen Abhängigkeiten führen zur Synchronisierung von Geschäftsprozessen [Schuetz, 2015, S. 76 ff.]. Synchronisationsabhängigkeiten wirken sich unterschiedlich auf den Lebenszyklus von MBAs aus, da sie Vor- oder Nachbedingungen von Zustandsübergängen sein können. Das Auslösen

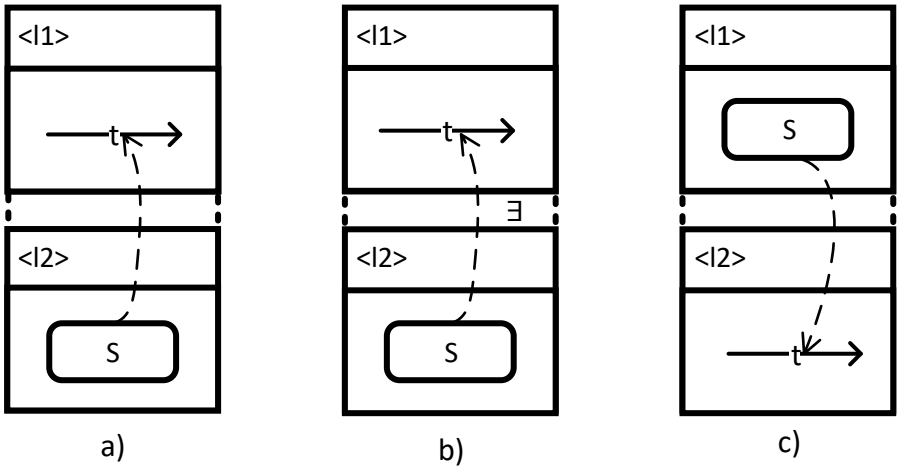

Abbildung 4: Synchronisationsbeziehungen Vorbedingungen (nach [Schuetz, 2015, S. 85 ff.])

einer Transition mit Vorbedingung setzt voraus, dass die entsprechende Bedingung von anderen Ebenen bzw. MBAs erfüllt ist, die Transition an sich hat keine Auswirkungen auf deren Datenobjekte. Davon unterscheidet sich eine Transition mit Nachbedingung dahingehend, dass ein Auslösen des Zustandsübergangs Änderungen in anderen Ebenen oder MBAs mit sich bringt. Vertikale Synchronisationsabhängigkeiten beziehen sich entweder auf Attribute eines MBAs (*attribute*), den aktuellen Zustand (*state*) oder erzeugen Konkretisierungen (*concretization*). Das Analysemodul ermöglicht die Auswertung von Vorbedingungen in Form von vertikalen Synchronisationsabhängigkeiten über Zustände. Tabelle 1 beschreibt die unterstützten Prädikate. Diese werden in den Bedingungen der betroffenen Transitionen angegeben und vom SCXML-Interpreter ausgewertet.

Abbildung 4 zeigt die graphische Notation von Synchronisationsbeziehungen nach [Schuetz, 2015, S. 85 ff.]. Gestrichelte Linien zeigen Abhängigkeiten an, wobei ein Pfeil zu einer Transition eine Vorbindung kennzeichnet. Das MBA von a) in Abbildung 4 beinhaltet das Prädikat `everyDescendantAtLevelIsInState` und b) `someDescendantAtLevelIsInState`. Letzteres wird über den Existenzquantor („∃“) markiert. Das MBA in c) besitzt eine Transition, dessen Vorbedingung aus dem Prädikat `ancestorAtLevelIsInState` besteht. Dieses wird dadurch erkannt, dass die Pfeilrichtung von der oberen zur unteren Ebene zeigt. Die zwei übrigen Prädikate, `isDescendantAtLevelInState` und `isAncestorAtLevelInState`, beziehen sich jeweils auf ein einzelnes MBA und werden daher nicht graphisch dargestellt.

Ähnlich wie Zustände oder Transitionen können Synchronisationsbeziehungen im Zuge von Konkretisierungen ebenfalls spezialisiert werden. Dies geschieht durch

Tabelle 1: Prädikate für vertikale Synchronisationsbeziehungen über Zustände [Schuetz, 2015, S. 83 f.]

Prädikat	**Parameter**	**Beschreibung**
everyDescendantAtLevelIsInState	MBA Ebene Zustand	Jede Konkretisierung des MBAs der angegebenen Ebene muss sich in dem festgelegten Zustand befinden.
someDescendantAtLevelIsInState	MBA Ebene Zustand	Zumindest eine Konkretisierung des MBAs der angegebenen Ebene muss sich in dem festgelegten Zustand befinden.
isDescendantAtLevelInState	MBA MBA Ebene Zustand	Das zweite MBA muss eine Konkretisierung des ersten MBAs der angegebenen Ebene sein und sich in dem festgelegten Zustand befinden.
ancestorAtLevelIsInState	MBA Ebene Zustand	Der Vorgänger des MBAs der angegebenen Ebene muss sich in dem festgelegten Zustand befinden.
isAncestorAtLevelInState	MBA MBA Ebene Zustand	Das zweite MBA ist ein Vorgänger des ersten MBAs der angegebenen Ebene und muss sich in dem festgelegten Zustand befinden.

Verstärken der Transitionsbedingungen, indem beispielsweise der im Prädikat angegebene Zustand durch einen seiner Subzustände ersetzt wird [Schuetz, 2015, S. 107 ff.]. Analog zur Spezialisierung von Transitionsbedingungen kann das umgesetzt werden, indem verschiedene Prädikate mithilfe einer UND-Verknüpfung (vgl. 2.1.1 Konkretisierungshierarchien) aneinandergereiht werden. Das vorliegende Analysemodul wurde für die Auswertung von Synchronisationsbeziehungen entwickelt, deren zugehörige Transitionsbedingung lediglich aus einem Prädikat besteht. Die Spezialisierung von Synchronisationsabhängigkeiten wird nicht unterstützt, um dies zu ermöglichen müssten aufwendigere Funktionen zum Parsen der Transitionsbedingungen implementiert werden. Die aktuelle Umsetzung dieser Funktionen ist in Abschnitt 4.2.2 beschrieben.

2.2 XML-Repräsentation mehrstufiger Geschäftsprozesse

Dieser Abschnitt befasst sich mit der logischen, XML-basierten Repräsentation von MBAs (siehe auch [Schuetz, 2015, S. 131 ff.]). Zu Beginn werden XML (*extensible markup language*) und die beiden Abfragesprachen XPath und XQuery erläutert. Nach einer Beschreibung der generellen Struktur von MBAs folgt eine Erklärung der Darstellung von den Prozessmodellen und der vom Analysemodul unterstützen Prädikate für die Synchronisation von Prozessen. Der Abschnitt schließt mit der Aufzeichnung der durchlaufenen Lebenszyklen von MBAs.

2.2.1 XML, XPath und XQuery

XML [W3C, 2008] ist eine vom W3C empfohlene Auszeichnungssprache, mit der hierarchisch strukturierte Dokumente erstellt werden können. Ein XML-Dokument besteht immer aus einem oder mehreren Elementen. Diese sind durch Start- und End-Tags gekennzeichnet. Der Teil zwischen diesen beiden Tags wird als Inhalt des Elements bezeichnet. Damit ein Dokument wohlgeformt ist, muss es genau einen Wurzelknoten besitzen. Quellcode 2 zeigt ein beispielhaftes XML-Dokument. Das `mba`-Element stellt den Wurzelknoten dar. Darunter befinden sich die beiden Kindknoten `topLevel` und `concretizations`. Letzterer besitzt keinen Inhalt und kann daher auch in Kurzschreibweise von nur einem Tag dargestellt werden. Neben dem Inhalt können Elemente auch über Attribute verfügen, welche im Start-Tag angegeben werden. Im Beispiel von Quellcode 2 besitzt das `topLevel`-Element das Attribut `name` mit dem Werte „strategicInsurance“.

In diesem Modul werden die beiden Abfragesprachen XPath [Clark und DeRose, 2015] und XQuery [W3C, 2014b] verwendet. Erstere ermöglicht das Abfragen von Teilen eines XML-Dokuments durch die Angabe des Pfades. Das Dokument wird dabei als Baum von Knoten angesehen. Quellcode 3 zeigt einen beispielhaften XPath-Ausdruck. Dadurch werden alle (`"//"`) `childLevel`-Kindelemente des

Quellcode 2: XML-Dokument

```
<mba>
  <topLevel name="strategicInsurance">
    <elements>
      ...
    </elements>
    <childLevel name="tacticalInsurance">
      ...
    </childLevel>
  </topLevel>
  <concretizations />
</mba>
```

Quellcode 3: XPath-Ausdruck

```
/mba//childLevel[@name="tacticalInsurance"]
```

Wurzelknotens `mba` selektiert, die über ein Attribut `name` mit dem Wert „tactical-Insurance“ verfügen. XQuery basiert auf XPath und erweitert ihre Funktionalität. Daraus folgt, dass jeder gültige XPath-Ausdruck eine gültige Anweisung in XQuery ist.

Neben Pfadangaben können in XQuery FLWOR-Ausdrücke (`for`, `let`, `where`, `order by` und `return`) verwendet werden, ein Beispiel dazu ist in Quellcode 4 dargestellt. Das Schlüsselwort `for` wird verwendet, um über jedes spezifizierte Element aus der Sequenz, die nach dem Schlüsselwort `in` angegeben wurde, zu iterieren. Das jeweilige Element wird dabei an die angegebene Variable gebunden. Mittels `let` können Variablen deklariert werden, `where` ermöglicht das Filtern des Ergebnisses. Durch Verwendung der Anweisung `order by` kann die Ergebnissequenz noch sortiert werden, bevor sie mit `return` zurückgeliefert wird.

XPath und XQuery verwenden das gleiche Datenmodell, welches alle gültigen Ausdrücke definiert. Die nachstehenden Absätze gehen auf ausgewählte Aspekte davon ein und erläutern diese.

Quellcode 4: FLWOR-Ausdruck

```
for $conc in $mba//mba
let $childLevel := $conc//childLevel[@name="operationalInsurance"]
where not(empty($childLevel))
order by $conc/@name
return $conc
```

In einem XML-Dokument besitzt jeder Knoten eine eindeutige Identität. Diese wird nicht manuell vergeben und ist unabhängig vom Inhalt des Knotens. Folglich können zwei inhaltlich gleiche Knoten unterschiedliche Identitäten besitzen. Diese Eigenschaft wird als *node identity* [W3C, 2015, #node-identity] bezeichnet. Werden Knoten zu einer Sequenz hinzugefügt, behalten sie ihre *node identity*. Die Gleichheit von Knoten aufgrund der *node identity* kann mithilfe des Schlüsselworts `is` geprüft werden.

Ein weitere im Datenmodell festgelegt Eigenschaft ist die Reihenfolge, in der von Anweisungen adressierte Knoten abgearbeitet werden. Sie wird als *document order* [W3C, 2015, #document-order] bezeichnet und verändert sich nicht während der Verarbeitung eines XML-Dokuments. Zuerst wird der Wurzelknoten des Dokuments besucht. Des Weiteren werden Elternknoten stets vor ihren Kindknoten bzw. übrigen Nachfolgern verarbeitet. Bei einem Knoten erfolgt zuerst die Verarbeitung von Namespaces und anschließend die der Attribute. Die Reihenfolge, in der Kindknoten abgearbeitet werden, entspricht jener, in der sie im Dokument auftreten. Darüber hinaus gilt dabei, dass Kindknoten eines Elements jeweils vor weiteren Geschwisterknoten abgearbeitet werden.

Neben diesen beiden Grundsätzen ist die im Datenmodell definierte Map-Datenstruktur hervorzuheben, da sie vom Analysemodul zur Schleifenerkennung verwendet werden. Ein Map-Item ist eine Sammlung von Key-Value-Paaren. Die Keys unterliegen der Einschränkung, dass sie innerhalb einer Map eindeutig sein und einem atomaren Datentyp entsprechen müssen. Jeder Key dient zur Identifikation des zugehörigen Value-Eintrags, der keinerlei inhaltlichen Einschränkungen unterliegt. Der Zugriff auf Einträge der Map erfolgt über entsprechende Funktionen, wie z.B. `map-key`, `map-contains` oder `map-get`.

XPath und XQuery besitzen eine Vielzahl an Built-In Funktionen [W3C, 2014a]. An diesem Punkt wird die vom Analysemodul verwendete Funktion `fold-left` [W3C, 2014a, #func-fold-left] erläutert. Auf weitere Built-In Funktionen wird bei der Implementierungsbeschreibung eingegangen. Die Funktion `fold-left` ermöglicht die sequentielle Verarbeitung von Knoten einer Sequenz von links nach rechts, wobei auf jedes Element eine Funktion angewandt wird. Die Resultate werden in einer Variable gesammelt, die bei den Funktionen jeweils mitgegeben wird. Quellcode 5 zeigt die Signatur von `fold-left`. Der erste Parameter (`$sec`) beinhaltet die zu verarbeitende Sequenz, der zweite (`$zero`) den Startwert des kumulierten Ergebnisses. Der letzte Parameter ist die Funktion, welche auf alle Elemente der Sequenz angewandt wird. Ihr erster Parameter repräsentiert das kumulierte Ergebnis und der zweite das aktuell zu verarbeitende Element der Sequenz. Um eine Sequenz von rechts nach links zu verarbeiten, muss `fold-right` verwendet werden, die analog zu `fold-left` funktioniert.

Quellcode 5: Signatur der Built-In Funktion `fold-left`

```
1 fn:fold-left(
2   $seq as item()*,
3   $zero  as item()*,
4   $f   as function(item()*, item()) as item()*
5 ) as item()*
```

2.2.2 Multilevel Business Artifacts

Die von [Schuetz, 2015, S. 131 ff.] vorgestellte Repräsentation von MBAs baut auf der W3C Empfehlung von *State Chart XML* (SCXML) [Barnett et al., 2015] auf, einer XML-basierten Sprache für Zustandsautomaten. Dies begünstigt zum einen die automatisierte Ausführung von MBAs und zum anderen ermöglicht die Flexibilität von SCXML die Unterstützung der hetero-homogenen Modellierung. Die Struktur des XML-Dokumentes lässt sich vom Aufbau von Konkretisierungshierarchien ableiten. Für einfache Hierarchien, wie sie vom Analysemodul unterstützt werden, bietet sich eine verschachtelte Repräsentation von MBAs und ihren Konkretisierungen an. Jedes MBA wird zu einem `mba`-Element mit dem Attribut `name` für seine Bezeichnung. Diese muss innerhalb einer Konkretisierungshierarchie eindeutig sein. Ein `mba`-Element besitzt immer genau einen Kindknoten `topLevel`, welcher die oberste Ebene des MBAs darstellt und ein Attribut `name` besitzt, welches die Bezeichnung der Ebene enthält. Unter diesem befindet sich ein `element`-Element, welches das Daten- und Prozessmodell der Ebene nach dem SCXML-Standard enthält. Ein `topLevel`-Element kann über einen Kindknoten `childLevel` verfügen, dieser repräsentiert die dem Top-Level untergeordnete Ebene und enthält das Daten- und Prozessmodell ebenfalls in einem SCXML-Modell. Des Weiteren kann ein `childLevel`-Element wiederum einen `childLevel`-Kindknoten besitzen. Die Konkretisierungen eines MBAs werden als Kinder eines `concretizations`-Elements festgehalten, das sich nach dem `topLevel`-Element befindet. Sie besitzen die gleiche Struktur wie das übergeordnete MBA.

Quellcode 6 zeigt die logische Darstellung einer Konkretisierungshierarchie. Im Beispiel ist „MyInsuranceCompany“ das abstrakteste MBA der Hierarchie und wird in der XML-Darstellung zum Wurzelelement übersetzt. Die oberste und somit instanziierte Ebene des MBAs trägt die Bezeichnung „strategicInsurance“. Darin befindet sich ein `childLevel`-Element, das die nächste hierarchische Ebene „tacticalInsurance“ darstellt und mit „operationalInsurance“ wiederum eine nachfolgende Ebene enthält. Das MBA „MyInsuranceCompany“ verfügt über die Konkretisierungen „MyCarInsuranceCompany“ und „MyHouseholdInsuranceCompany“, die entsprechend den Konkretisierungsregeln (vgl. 2.1.1 Konkretisierungshierarchien) mit „tacticalInsurance“ die zweite Ebene von „MyInsuranceCompany“ instan-

Quellcode 6: XML-Darstellung einer Konkretisierungshierarchie

```
1  <mba name="MyInsuranceCompany"
2      xmlns="http://www.dke.jku.at/MBA"
3      xmlns:sc="http://www.w3.org/2005/07/scxml">
4    <topLevel name="strategicInsurance">
5      <elements>
6        ...
7      </elements>
8      <childLevel name="tacticalInsurance">
9        <elements>
10         ...
11       </elements>
12       <childLevel name="operationalInsurance">
13         <elements>
14           ...
15         </elements>
16       </childLevel>
17     </childLevel>
18   </topLevel>
19   <concretizations>
20     <mba name="MyCarInsuranceCompany"
21         xmlns="http://www.dke.jku.at/MBA"
22         xmlns:sc="http://www.w3.org/2005/07/scxml">
23       <topLevel name="tacticalInsurance">
24         ...
25       </topLevel>
26       <concretizations>
27         ...
28       </concretizations>
29     </mba>
30     <mba name="MyHouseholdInsuranceCompany"
31         xmlns="http://www.dke.jku.at/MBA"
32         xmlns:sc="http://www.w3.org/2005/07/scxml">
33       <topLevel name="tacticalInsurance">
34         ...
35       </topLevel>
36       <concretizations>
37         ...
38       </concretizations>
39     </mba>
40   </concretizations>
41 </mba>
```

ziieren. Jedes dieser `mba`-Elemente besitzt neben dem `topLevel` den Kindknoten `concretizations`, unter dem weitere Konkretisierungen gespeichert werden können.

Das Prozessmodell des Top-Levels des MBAs aus Quellcode 6 wird in Quellcode 7 gezeigt. In einem SCXML-Dokument wird ein Zustandsautomat immer von einem `scxml`-Element gekapselt, zu sehen in Zeile 6. Dieses verfügt über das Attribut `name`, dessen Wert der Bezeichnung des MBAs entspricht. Der eigentliche Zustandsautomat besteht aus dem Daten- und Prozessmodell. Letzteres setzt sich aus einer Sequenz von Zuständen (`state`, `parallel`, `final` oder `initial`) zusammen und wird in Abschnitt 2.2.3 genauer erläutert (siehe auch [Barnett et al., 2015]).

Daten eines MBAs werden als Kinder von `datamodel` gespeichert, dabei entspricht ein `data`-Element einem Datenelement. Im Beispiel von Quellcode 7 wird in Zeile 8 ein Element für die Beschreibung des MBAs angelegt. Die Bezeichnung (`id`-Attribut) von benutzerdefinierten Daten darf nicht mit einem Unterstrich („_“) beginnen, da auf diese Art Systemvariablen gekennzeichnet werden. Das Analysemodul verwendet für die Auswertung von MBAs Daten, die in dem `data`-Element mit der `id` „_x“ gespeichert sind. Es beinhaltet plattformabhängige Systemdaten [Barnett et al., 2015, #SystemVariables], unter anderem die Bezeichnung der verwendeten XML-Datenbank (`db`), der Konkretisierungshierarchie (`collection`), des MBAs (`mba`), des aktuell aktiven Zustands (`currentStatus`) und Event-Logs (`log`) [Schuetz, 2015, S. 137 ff.]. Letzteres dient als Basis für die Analyse von MBAs und wird in 2.2.5 Logging beschrieben.

2.2.3 Prozessmodelle

Prozessmodelle von MBAs werden unter Verwendung von SCXML [Barnett et al., 2015] als Zustandsautomaten modelliert. Grundlegende Elemente sind `state` und `transition`. Ersteres definiert einen Zustand, dessen Bezeichnung in dem `id`-Attribut gespeichert wird. Ein `state`-Element kann der Elternknoten eines weiteren `state`-Elements sein, damit wird die Verwendung von zusammengesetzten Zuständen ermöglicht, zu sehen in Quellcode 8. Ein Zustand, der über keine Subzustände verfügt, wird als atomar bezeichnet. Im Beispiel sind „A“ und „B“ zusammengesetzte und „C“ bzw. „D“ atomare Zustände. Zustandsübergänge werden mithilfe von `transition`-Elementen festgelegt. Dieses kann unter anderem die drei optionalen Attribute `event`, `cond` und `target` besitzen. Das erste beschreibt das Ereignis, das die Transition auslöst, das zweite eine Bedingung die dafür erfüllt sein muss und das letzte den Zielzustand der Transition. Dadurch ist es beispielsweise möglich, dass Transitionen keine Zustandsveränderung hervorrufen oder ohne Ereignis ausgelöst werden können.

Quellcode 7: XML-Darstellung des Daten- und Prozessmodells eines MBAs

```
1  <mba name="MyInsuranceCompany"
2      xmlns="http://www.dke.jku.at/MBA"
3      xmlns:sc="http://www.w3.org/2005/07/scxml">
4    <topLevel name="strategicInsurance">
5      <elements>
6        <sc:scxml name="StrategicInsurance">
7          <sc:datamodel>
8            <sc:data id="description">
9              Insurance Company
10           </sc:data>
11           <sc:data id="_event">
12             ...
13           </sc:data>
14           <sc:data id="_x">
15             <db xmlns="">myMBAse</db>
16             <collection xmlns="">MyInsuranceCompany</collection>
17             <mba xmlns="">MyInsuranceCompany</mba>
18             <currentStatus xmlns="">
19               <state ref="Working"/>
20             </currentStatus>
21             <xes:log xmlns:xes="http://www.xes-standard.org /">
22               ...
23             </xes:log>
24           </sc:data>
25         </sc:datamodel>
26         <sc:initial>
27           <sc:transition target="Working" />
28         </sc:initial>
29         <sc:state id="Working">
30           <sc:transition event="handleIncident" />
31           <sc:transition event="closeCompany" target="Closed" />
32         </sc:state>
33         <sc:state id="Closed" mba:isArchiveState="true" />
34       </sc:scxml>
35     </elements>
36     <childLevel name="tacticalInsurance">
37       ...
38     </childLevel>
39   </topLevel>
40   <concretizations>
41     ...
42   </concretizations>
43 </mba>
```

Wird eine Transition mit einem Zielzustand ausgelöst, führt das dazu, dass der aktuelle Zustand verlassen und der Zielzustand betreten wird. Ist dieser ein Teil eines zusammengesetzten Zustands, werden seine übergeordneten Zustände ebenfalls aktiv. Wenn ein zusammengesetzter Zustand aktiv ist, muss immer genau eines seiner Kinder aktiv sein. Das kann z.B. über Initialzustände (`initial`) oder das `initial`-Attribut sichergestellt werden. Quellcode 8 zeigt einen zusammengesetzten Zustand mit einem Initialzustand (`initial`). Dieser führt dazu, dass beim Betreten von „A“ der Zustand „B“ aktiv wird, zu sehen in den Zeilen 2 bis 4. Ein `initial`-Element besitzt keinerlei Attribute und hat immer genau einen Kindknoten `transition`. Diese Transition darf kein Ereignis (`event`) spezifizieren und muss über einen Zielzustand verfügen (`target`). Der Initialzustand wird automatisch aktiv wenn der übergeordnete Zustand betreten wurde. Der Zustand „B“ definiert „C“ mithilfe des `initial`-Attributs als seinen Initialzustand. Dadurch ist sichergestellt, dass wenn eine Transition auf „A“ ausgelöst wird, genau ein Kind eines jeden zusammengesetzten Zustands aktiv ist.

Falls bei zusammengesetzten Zuständen Initialzustände fehlen sollten, sieht die SCXML-Empfehlung des W3C vor, dass beim Betreten des Elternzustands der erste Kindzustand in *document order* aktiv wird. Das vorliegende Analysemodul hat diesen Punkt nicht umgesetzt, d.h. für die korrekte Ausführung der Analyse muss sichergestellt werden, dass entsprechende Initialzustände vergeben bzw. Transitionen so definiert werden, dass die Prüfung der *document order* nicht benötigt wird. Es wird vom Analysemodul diesbezüglich keine Prüfung des Prozessmodells vorgenommen, diese Verantwortung obliegt den Benutzerinnen und Benutzern.

Damit die Lebenszyklen von MBAs vollständig analysiert werden können, müssen die Prozessmodelle mit einem `initial`-Element begonnen werden. Dieses muss eine ereignislose Transition beinhalten die auf den ersten aktiven Zustand des MBAs zeigt.

Endzustände können in SCXML-Zustandsautomaten mit dem `final`-Element umgesetzt werden um zu signalisieren, dass etwas abgeschlossen wurde. Dieses verfügt über ein `id`-Attribut und darf keine untergeordneten Zustände besitzen. Wird der Zustand betreten, erzeugt der SCXML-Prozessor das Ereignis „`done.state.`*`id`*“, das beispielsweise verwendet werden kann um weitere Transitionen auszulösen.

Parallele Zustände werden mithilfe des `parallel`-Elements modelliert. Das vorliegende Modul wurde für die in Quellcode 9 dargestellte Verwendung von parallelen Zuständen konzipiert, bei der parallele Pfade am Ende wieder zusammengeführt werden. Diese hat den Vorteil, dass ein `parallel`-Element nur dann verlassen werden kann, wenn alle Subprozesse abgeschlossen wurden. Würde man auf die Transitionsbedingung verzichten, kann eine vollständige Ausführung der Subzustände nicht sichergestellt werden. Das `parallel`-Element hat zwei oder

Quellcode 8: Zusammengesetzter Zustand

```
<sc:state id="A">
  <sc:initial>
    <sc:transition target="B" />
  </sc:initial>
  <sc:state id="B" initial="C">
    <sc:state id="C">
      <sc:transition event="t" target="D" />
    </sc:state>
  </sc:state>
  <sc:state id="D">
    <sc:transition event="t" target="E" />
  </sc:state>
</sc:state>
```

mehrere Kindzustände, die beim Betreten des `parallel`-Elements bzw. eines seiner Subzustände gleichzeitig aktiv werden. Die parallelen Prozesse befinden sich in den Subzuständen und jeder dieser Subprozesse wird durch ein `final`-Element beendet. Das `parallel`-Element wird durch eine Transition verlassen, die ausgelöst wird wenn sich alle Subprozesse in einem finalen Zustand befinden.

Quellcode 9: Parallele Zustände

```
<sc:parallel id="P">
  <sc:state id="A" initial="A'">
    <sc:state id="A'">
      <sc:transition event="t2" target="A_f"/>
    </sc:state>
    <sc:final id="A_f"/>
  </sc:state>
  <sc:state id="B" initial="B'">
    <sc:state id="B'">
      <sc:transition event="t3" target="B_f"/>
    </sc:state>
    <sc:final id="B_f"/>
  </sc:state>
  <sc:transition target="C" event="done.state.P"/>
</sc:parallel>
```

2.2.4 Prädikate für Synchronisationsabhängigkeiten

Abhängigkeiten zwischen Prozessmodellen verschiedener Ebenen für deren Synchronisierung (vgl. 2.1.2 Synchronisation von mehrstufigen Prozessmodellen) sind in Form von Prädikaten umgesetzt [Schuetz, 2015, S. 145 ff.]. Diese werden in den Bedingungen (`cond`-Attribut) von Transitionen verwendet und vom SCXML-

Interpreter ausgewertet. Eine beispielhafte Transition ist in Quellcode 10 definiert. Der Zustandsübergang wird verwendet, wenn das Ereignis „t“ auftritt und sich jede Konkretisierung des MBAs auf der Ebene „l2“ im Zustand „B“ befindet. Das Prädikat wird vom SCXML-Interpreter im Kontext des entsprechenden MBAs ausgewertet, daher muss dieses nicht explizit als Parameter angegeben werden.

Quellcode 10: Transition mit Vorbedingung

```
1 <sc:transition
2     event="t"
3     target="A"
4     cond="$_everyDescendantAtLevelIsInState('l2', 'B')"
5 />
```

Tabelle 2 listet die XQuery-Funktionen für vertikale Synchronisationsbeziehungen über Zustände mit ihren Parametern auf [Schuetz, 2015, S. 150 ff.]. Die Funktion `$_everyDescendantAtLevelIsInState` wird vom SCXML-Interpreter zu „true“ evaluiert, wenn sich jede Konkretisierung des MBAs der Ebene `$level` im Zustand mit der Bezeichnung `$stateId` befindet. Die zweite Funktion, `$_someDescendantAtLevelIsInState`, liefert „true“, wenn der Zustand `$stateId` bei zumindest einer Konkretisierung der Ebene `$level` aktiv ist. Als dritte Funktion ermöglicht `$_isDescendantAtLevelInState` die Spezifizierung einer bestimmten Konkretisierung, die für den ersten Parameter `$obj` übergeben wird. Sie wird mithilfe eines XPath-/XQuery-Ausdrucks angegeben und muss das MBA, in dem die Synchronisationsbeziehung definiert wird, auf der angegebenen Ebene (`$level`) konkretisieren. Die Funktion liefert „true“, wenn sich das angegebene MBA `$obj` in dem Zustand `$stateId` befindet. Eine Prüfung auf den aktuellen Zustand des Vorgängers eines MBAs kann mithilfe von `$_ancestorAtLevelIsInState` vorgenommen werden, diese Funktion liefert „true“, wenn sich der Vorgänger der Ebene `$level` im Zustand mit der Bezeichnung `$stateId` befindet. Analog zu `$_isDescendantAtLevelInState` funktioniert `$_isAncestorAtLevelInState`: für den ersten Parameter `$obj` wird ein MBA übergeben, das ein Vorgänger des entsprechenden MBAs auf der angegebenen Ebene (`$level`) sein muss.

Das Analysemodul unterstützt die Auswertung von den in Tabelle 2 aufgelisteten Prädikaten zur Modellierung von Synchronisationsabhängigkeiten zwischen Prozessmodellen. Vorbedingungen in Transitionen sind dabei der Einschränkung unterworfen, dass sie immer nur aus einem Prädikat bestehen dürfen. Des Weiteren wird zur Auswertung des `$obj` Parameters bei den Funktionen `$_isDescendantAtLevelInState` und `$_isAncestorAtLevelInState` die Built-In Funktion `eval` verwendet [BaseX, 2016, #xquery:eval]. Diese evaluiert XPath-/XQuery-Ausdrücke in einem eigenen Kontext, daher muss der Parameter `$obj` jeweils in einer kontextunabhängigen Form angegeben werden.

Tabelle 2: XQuery-Funktionen für vertikale Synchronisationsbeziehungen über Zustände (nach [Schuetz, 2015, S. 150 ff.])

Funktion	**Parameter**
$_everyDescendantAtLevelIsInState	$level $stateId
$_someDescendantAtLevelIsInState	$level $stateId
$_isDescendantAtLevelInState	$obj $level $stateId
$_ancestorAtLevelIsInState	$level $stateId
$_isAncestorAtLevelInState	$obj $level $stateId

2.2.5 Logging

Um die automatische Ausführung von Prozessen analysieren zu können, müssen Ereignisse in einem Log gespeichert werden [Dumas et al., 2013, S. 353 ff.]. MBAs besitzen ein Event-Log, welches bei der automatisierten Ausführung erstellt wird [Schuetz, 2015, S. 188 ff.]. Dieses Event-Log beinhaltet sämtliche aufgetretenen Ereignisse mit deren relevanten Daten, darunter Zeiten und Zustandsübergänge. Mit diesen Informationen kann der Lebenszyklus eines MBAs rekonstruiert werden und des Weiteren ermöglichen sie die Berechnung von Durchlaufzeiten [Schuetz, 2015, S. 188 ff.]. Quellcode 11 zeigt ein beispielhaftes Event-Log. Der darin dargestellte Prozess wurde am 1. Jänner 2016 um 07:30 gestartet. Der erste aktive Zustand war „ChooseProducts“, der Initialzustand des SCXML-Prozessmodells „TacticalInsurance“. Danach trat um 07:37 das Ereignis „collectInformation“ auf. Um 07:50 erfolgte durch das Ereignis „startDevelopment“ ein Zustandsübergang von „ChooseProducts“ nach „DevelopProducts“.

Event-Logs werden im Datenmodell des MBAs unter den Systemvariablen („_x“) gespeichert. Unter dem `log`-Element befindet sich ein `trace`-Element, das eine Sequenz von Ereignissen (`event`) enthält. Es gibt genau ein `event`-Element für jedes Auftreten einer Transition im Lebenszyklus des MBAs. Dabei ist es irrelevant, ob ein Zustandsübergang erfolgte oder die Transition von einem Ereignis ausgelöst wurde.

Die Daten der Ereignisse werden in Key-Value-Paaren gespeichert und zur Aufzeichnung wird der XES-Standard (e**X**tensible **E**vent **S**tream) [Günther und Verbeek, 2014] verwendet. Dieser beschreibt eine XML-basierte Notation von Event-

Quellcode 11: Event-Log

```
<sc:data id="_x">
  ...
  <xes:log xmlns:xes="http://www.xes-standard.org/">
    <xes:trace>
      <xes:event>
        <xes:date key="time:timestamp"
          value="2016-01-01T07:30:00.000+02:00"/>
        <xes:string key="sc:initial" value="TacticalInsurance"/>
        <xes:string key="sc:target" value="ChooseProducts"/>
      </xes:event>
      <xes:event>
        <xes:date key="time:timestamp"
          value="2016-01-01T07:37:00.000+02:00"/>
        <xes:string key="sc:state" value="ChooseProducts"/>
        <xes:string key="concept:name" value="collectInformation"/>
        <xes:string key="sc:event" value="collectInformation"/>
      </xes:event>
      <xes:event>
        <xes:date key="time:timestamp"
          value="2016-01-01T07:50:00.000+02:00"/>
        <xes:string key="sc:state" value="ChooseProducts"/>
        <xes:string key="concept:name" value="startDevelopment"/>
        <xes:string key="sc:event" value="startDevelopment"/>
        <xes:string key="sc:target" value="DevelopProducts"/>
      </xes:event>
    </xes:trace>
  </xes:log>
</sc:data>
```

Logs. Um Ereignisse von MBAs vollständig abbilden zu können, wurden dem Standard weitere Element hinzugefügt (vgl. Tabelle 3) [Schuetz, 2015, S. 188 ff.]. Es ist anzumerken, dass `sc:state` und `sc:initial` nicht gleichzeitig unter demselben `event`-Element vorkommen dürfen, allerdings muss jedes `event`-Element eines davon enthalten. Man beachte den Unterschied zwischen den Einträgen `concept:name` und `sc:event`. Letztere beinhaltet einen Deskriptor für Ereignisse, die eine Transition auslösen können, während der erste das tatsächlich aufgetretene Ereignis festhält (siehe auch [Barnett et al., 2015, #events]). Deskriptoren ermöglichen das genauere Spezifizieren von Ereignissen mithilfe des Punkt-Operators, so entspricht beispielsweise das Ereignis „event.1“ dem Deskriptor „event“.

Um die Analyse hinsichtlich der Durchlaufzeiten zu vereinfachen wird aus dem Event-Log ein State-Log generiert [Schuetz, 2015, S. 192 ff.]. Dieses beinhaltet jene Zustände, die im Lebenszyklus des MBAs aktiv waren sowie den jeweiligen Zeitraum. Quellcode 12 zeigt das zu Quellcode 11 zugehörige State-Log. Unter dem

Tabelle 3: Logging-Elemente (nach [Schuetz, 2015, S. 188 ff.])

Attribut	**Beschreibung**
`date:timestamp`	Zeitpunkt, zu dem das Ereignis aufgetreten ist.
`concept:name`	Ereignis, das die Transition tatsächlich ausgelöst hat.
`sc:event`	Deskriptor für Ereignisse, welche eine Transition auslösen könnten. Das Ereignis „event.1“ entspricht z.B. dem Deskriptor „event“.
`sc:cond`	Beschreibt die Bedingung der Transition.
`sc:state`	Bezeichnung des Ausgangszustands der Transition.
`sc:initial`	Wird verwendet, wenn sich die Transition in einem `sc:initial` Element befindet und hält die Bezeichnung des übergeordneten Zustands bzw. `scxml` Elements.
`sc:target`	Bezeichnung des Zielzustands der Transition.

Quellcode 12: State-Log

```
1 <stateLog>
2   <state ref="ChooseProducts"
3     from="2016-01-01T07:30:00.000+02:00"
4     until="2016-01-01T07:50:00.000+02:00"/>
5   <state ref="DevelopProducts"
6     from="2016-01-01T07:50:00.000+02:00"/>
7 </stateLog>
```

`stateLog`-Element befindet sich eine Sequenz von `state`-Elementen. Jedes davon muss die zwei Attribute `ref` und `from` enthalten. In `ref` wird die eindeutige Bezeichnung des Zustands gespeichert, `from` speichert den Zeitpunkt wann der Zustand aktiv wurde. Zu diesen zwei verpflichtenden kommt das optionale Attribut `until`. Es beschreibt den Zeitpunkt zu dem der Zustand verlassen wurde. Zustände die während der Generierung des State-Logs aktiv sind besitzen dieses Attribut nicht. Schleifen in einem Prozess können dazu führen, dass ein Zustand öfter als einmal aktiv wird. In diesem Fall enthält das State-Log mehrere Einträge für einen Zustand, daher kann ein Eintrag im State-Log nicht eindeutig über das `ref`-Attribut identifiziert werden.

2.3 Prozessanalyse

Dieser Abschnitt geht auf die Techniken ein, die vom Analysemodul zur Auswertung der Prozessmodelle verwendet werden. Auf die quantitative sowie qualitative Analyse von Prozessen folgt eine Erläuterung der Auswertung der Lebenszyklen von MBAs. Abschließend wird beschrieben, wie das Analysemodul Schleifen in Prozessen erkennt.

2.3.1 Quantitative Analyse

In diesem Abschnitt wird die vom Analysemodul verwendet Technik zur quantitativen Analyse von MBAs erläutert. Mithilfe der *Flow Analysis* [Dumas et al., 2013, S. 213 ff.] kann die Leistungsfähigkeit von Geschäftsprozessen anhand von Maßzahlen bewertet werden. Die obersten Ziele des Prozessmanagements sind Prozesse schneller, billiger und besser zu gestalten. Dazu kommt der Wunsch nach einer Steigerung der Flexibilität, d.h. Prozesse dahingehend zu optimieren, dass sich Änderungen in der Umwelt nicht bzw. weniger negativ auf seine Leistungsfähigkeit auswirken. An diesen Zielen orientieren sich ebenfalls die Maßzahlen der quantitativen Analyse, daher kann man diese in die vier Dimensionen Zeit, Kosten, Qualität und Flexibilität einteilen. Jede Dimension verfügt über Kennzahlen (engl. *key performance indicator*, KPI), die eindeutig berechnet werden können und Auskunft über die Leistungsfähigkeit des Prozesses geben (siehe auch [zur Mühlen und Shapiro, 2010]). Dazu kann eine Vielzahl an Aggregationsfunktionen wie beispielsweise Durchschnitt, Varianz oder Median verwendet werden. Darüber hinaus ist es durch das Bilden von Verhältnissen zwischen diese Metriken möglich, neue Erkenntnisse zu gewinnen.

Das vorliegende Analysemodul bewertet Lebenszyklen von MBAs anhand der durchschnittlichen Durchlaufzeit, nachfolgend vereinfachend auch als Durchlaufzeit bezeichnet. Das ist jene Dauer, die im Durchschnitt vom Beginn des Prozesses bis zu seinem Ende verstreicht [Dumas et al., 2013, S. 214]. Analog dazu kann die-

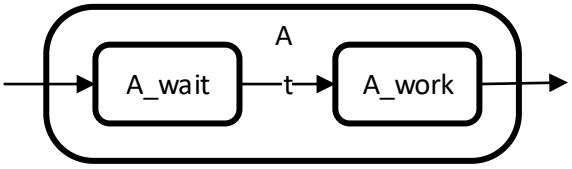

Abbildung 5: Beispiel für Modellierung von Wartezuständen

se Maßzahl neben gesamten Prozessen auch für Prozessabschnitte bzw. einzelne Zustände gebildet werden. Die Durchlaufzeit eines Prozessabschnitts ist die Dauer vom Eintritt in diesen Teilprozess, d.h. die Zeit zu dem der erste Zustand betreten wurde, bis zu dem Zeitpunkt, zu dem der letzte gewünschte Zustand aktiv wurde. Die Durchlaufzeit von Zuständen entspricht der Dauer, in der sie aktiv waren.

Für gewöhnlich setzt sich die Durchlaufzeit eines Prozesses aus der Bearbeitungs- und Wartezeit zusammen [Dumas et al., 2013, S. 224]. Erstere beschreibt jene Dauer, die zur tatsächlichen Erfüllung der Aufgabe benötigt wird, während letztere die inaktive Zeit zusammenfasst. Das ist beispielsweise die Dauer, in der ein Zustand verpflichtet ist, auf das Ergebnis eines Vorgängers zu warten und deswegen nicht fortfahren kann. Die Bearbeitungszeit wird auch als theoretische Durchlaufzeit bezeichnet, d.h. die Dauer die ein Prozess ohne Wartezeit benötigen würde. Die Effizienz eines Geschäftsprozesses wird berechnet indem die Bearbeitungszeit durch die gesamte Durchlaufzeit dividiert wird. Um den Detaillierungsgrad der Analyse von MBAs dahingehend zu erhöhen, müssen die Prozessmodelle um Wartezustände erweitert werden. Abbildung 5 zeigt wie ein Zustand verfeinert werden kann, um Bearbeitungs- bzw. Wartezeit berechnen zu können. Die Erweiterung des Zustands „A“ um die Subzustände „A_wait“ und „A_work“ ermöglicht eine genauere Aufschlüsselung der Durchlaufzeiten. Werden in einem Prozess alle Zustände dementsprechend verfeinert, ergibt sich die Bearbeitungszeit aus der Summe der Durchlaufzeiten aller Zustände mit dem Suffix „_work“.

Die *Flow Analysis* berechnet die Durchlaufzeit eines Prozesses durch aggregieren der Durchlaufzeiten der betroffenen Zustände. Abbildung 6 illustriert diese Vorgehensweise anhand eines sequentiellen Prozessmodells mit den beiden Zuständen „A“ und „B“. Die jeweilige Durchlaufzeit ist in Klammern neben der Zustandsbezeichnung notiert. Da das Prozessmodell keine Abzweigungen enthält und somit jeder Zustand betreten werden muss, ergibt sich die gesamte Durchlaufzeit aus der Summe der Durchlaufzeiten aller Zustände [Dumas et al., 2013, S. 219] , d.h. der Prozess dauert durchschnittlich 30 Zeiteinheiten.

Beinhaltet der zu untersuchende Prozess Verzweigungen, muss die Durchlaufzeit anders berechnet werden, da nicht alle Zustände des Prozesses betreten werden. Das Prozessmodell in Abbildung 7 besitzt alternative Pfade, da dem Zustand „A“ entweder „B“ oder „C“ folgen kann. Davon ist auch die Dauer des Prozesses

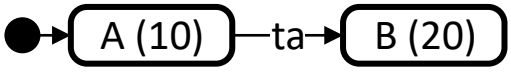

Abbildung 6: Sequentielles Prozessmodell

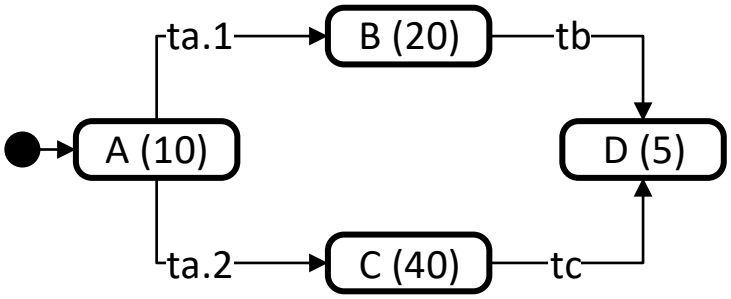

Abbildung 7: Prozessmodell mit alternativen Pfaden

abhängig, d.h. es müssen entweder 20 oder 40 Zeiteinheiten in die Durchlaufzeit miteinbezogen werden. Da dies von der Wahrscheinlichkeit, mit der die entsprechenden Transitionen „ta.1“ und „ta.2“ genommen werden, abhängig ist müssen die Durchlaufzeiten danach gewichtet werden [Dumas et al., 2013, S. 219f.]. Angenommen, die Transition „ta.1“ wird mit einer Wahrscheinlichkeit von 25%, d.h. in einem von vier Fällen, betreten und „ta.2“ zu 75%, dann beträgt die Durchlaufzeit der alternativen Pfade 35 (= 20*0,25 + 40*0,75) Zeiteinheiten. Um die gesamte Durchlaufzeit des Prozesses zu erhalten, muss noch die Dauer der Zustände „A“ und „D“ hinzugerechnet werden, demnach dauert der Prozess im Durchschnitt 50 (= 10 + 35 + 5) Zeiteinheiten.

Einen Sonderfall für alternative Pfade stellen Schleifen (*rework loops*) im Prozess dar, bei denen Abschnitte wiederholt werden. Ein Beispiel dafür zeigt Abbildung 8. Nach dem Zustand „B“ öffnet sich eine Verzweigung: mit einer Wahrscheinlichkeit von 50% folgt „D“, ansonsten wird eine Schleife betreten die über „C“ zurück zu „B“ führt. Die Wahrscheinlichkeit, dass der Zustand „B“ zwei Mal ausgeführt wird, beträgt 50%, für eine dritte Wiederholung dementsprechend 25% (= 0,5*0,5) usw. Allgemein formuliert beträgt die Wahrscheinlichkeit dieses Zustands $0{,}5^{\mathtt{n}}$, wobei `n` die Anzahl der Wiederholungen ist [Dumas et al., 2013, S. 222 f.]. Die nach dieser Wahrscheinlichkeit gewichtete Durchlaufzeit von „B“ beträgt für die erste Wiederholung 10, die zweite 5, usw. Werden die 50% mit der Variable `r` ersetzt, entspricht die Wahrscheinlichkeit, mit der die Durchlaufzeit eines Zustands in einer Schleife gewichtet werden muss, dem Ergebnis der folgenden geometrischen Reihe:

$$\sum_{n=0}^{\infty} r^n = \frac{1}{1-r}$$

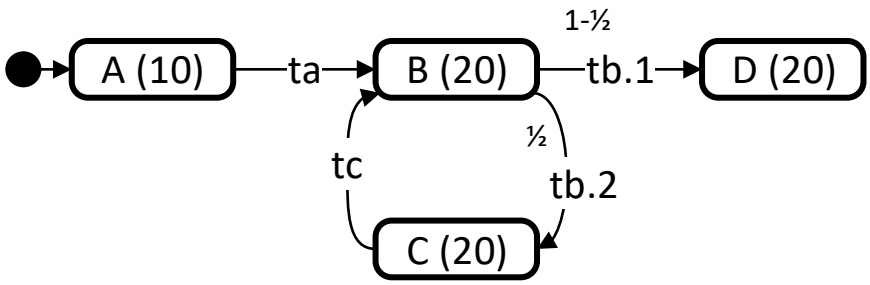

Abbildung 8: Prozessmodell mit einer Schleife

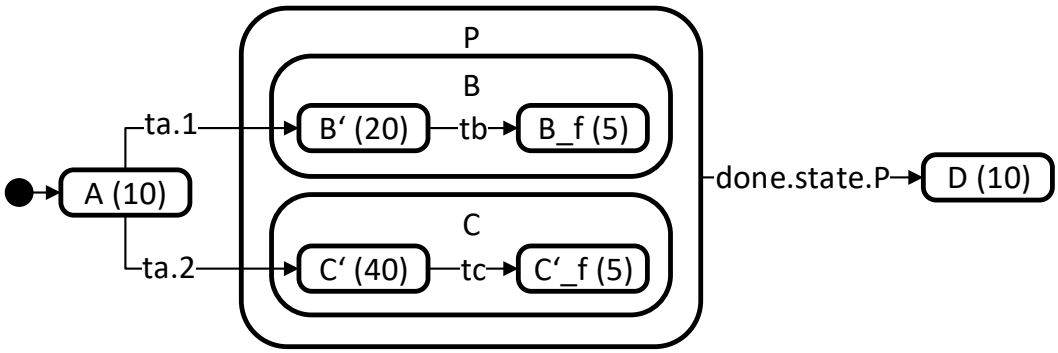

Abbildung 9: Prozessmodell mit `parallel`-Element

Für das Beispiel aus Abbildung 8 bedeutet das, dass der Zustand „B“ mit einer Wahrscheinlichkeit von 2 (= 1/(1 - 0,5)) auftritt. Daraus folgt, dass er mit einer Durchlaufzeit von 40 (= 2*20) Zeiteinheiten zu bewerten ist. Nachfolgend werden Prozessabschnitte mit Schleifen auch als *rework*-Blöcke bezeichnet.

Abbildung 9 zeigt ein Prozessmodell mit dem `parallel`-Element „P“. Das bedeutet für den Prozess, dass die Zustände „B“ und „C“ gleichzeitig betreten werden und beide abgeschlossen sein müssen, um den Zustand „D“ zu erreichen. Ähnlich wie bei alternativen Pfaden kann für die Durchlaufzeit nicht die Summe der Zeiten aller Zustände verwendet werden, da diese zur gleichen Zeit aktiv sind und demnach parallel abgearbeitet werden. Stattdessen entspricht die Durchlaufzeit von parallelen Zuständen der längsten Durchlaufzeit eines ihrer Subzustände, da stets alle abgeschlossen sein müssen um den Prozess fortsetzen zu können [Dumas et al., 2013, S. 220 f.]. Die daraus resultierende Durchlaufzeit für den parallelen Zustand im Prozessmodell von Abbildung 9 beträgt daher 45 (= max(25, 45)), die Durchlaufzeit des gesamten Prozesses 65 (= 10 + 45 + 10) Zeiteinheiten.

Im Analysemodul wird zwischen relativen und absoluten Transitionswahrscheinlichkeiten unterschieden. Die relative Wahrscheinlichkeit beschreibt, wie häufig eine Transition aktiv wurde, betrachtet von ihrem Ausgangszustand. Dazu wird

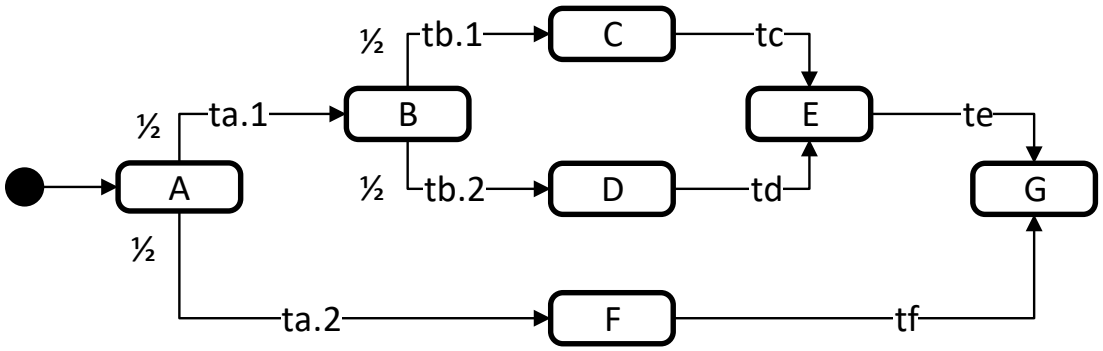

Abbildung 10: Vergleich relative und absolute Transitionswahrscheinlichkeiten

erfasst, wie häufig der Ausgangszustand verlassen und wie oft davon der Prozess über die entsprechende Transition fortgesetzt wurde. Der Quotient aus diesen beiden Werten ist die relative Wahrscheinlichkeit. Diese kann allerdings noch nicht für die Gewichtung von Durchlaufzeiten verwendet werden, da sie etwaige vorhergehende Transitionen nicht berücksichtigt. Abbildung 10 zeigt ein Prozessmodell, das diesen Umstand illustriert. Die relativen Transitionswahrscheinlichkeiten sind neben den entsprechenden Transitionen vermerkt. Der Zustand „C“ wird über die Transition mit dem Ereignis „tb.1“ erreicht, demnach folgt er in der Hälfte aller Fälle, in denen ihr Ausgangszustand („B“) aktiv war. Das entspricht einer relativen Wahrscheinlichkeit von 50% für den Zustand „C“. Eine Gewichtung der Durchlaufzeit von „C“ mit dieser Wahrscheinlichkeit wäre allerdings nicht korrekt, da sich vor seinem Vorgänger „B“ ebenfalls eine Verzweigung befindet. Um die absolute Wahrscheinlichkeit, mit der ein Zustand im Durchschnitt aktiv wird, zu erhalten müssen die Wahrscheinlichkeiten sowohl von allen direkten als auch indirekten vorhergehenden Transitionen berücksichtigt werden. In dem Beispiel von Abbildung 10 ergibt sich die absolute Wahrscheinlichkeit für den Zustand „C“ aus dem Produkt der relativen Wahrscheinlichkeiten aller Transitionen des Pfades, über den „C“ vom Initialzustand des Prozesses aus erreicht werden kann, d.h. „tb.1“ und „ta.1“. Das Ergebnis von 25% (= 0,5*0,5) bedeutet, dass der Zustand „C“ in durchschnittlich jeder vierten Prozessinstanz aktiv ist. Sollten mehrere Pfade zu einem Zustand führen, werden zuerst die Produkte der relativen Wahrscheinlichkeiten für jeden Pfad separat gebildet und anschließend aufsummiert.

Abbildung 11 illustriert die Berechnung von absoluten Transitionswahrscheinlichkeiten bei einem Prozessmodell mit Schleifen. Der Zustand „A“ wird von zwei Transitionen verlassen und einer dieser Zweige beinhaltet eine Schleife mit den Zuständen „B“, „C“ und „D“. Um Schleifen in Prozessmodellen erkennen zu können bedient sich das Analysemodul der Graphentheorie unter Verwendung von stark zusammenhängenden Komponenten (SZK, engl. *strongly connected components*

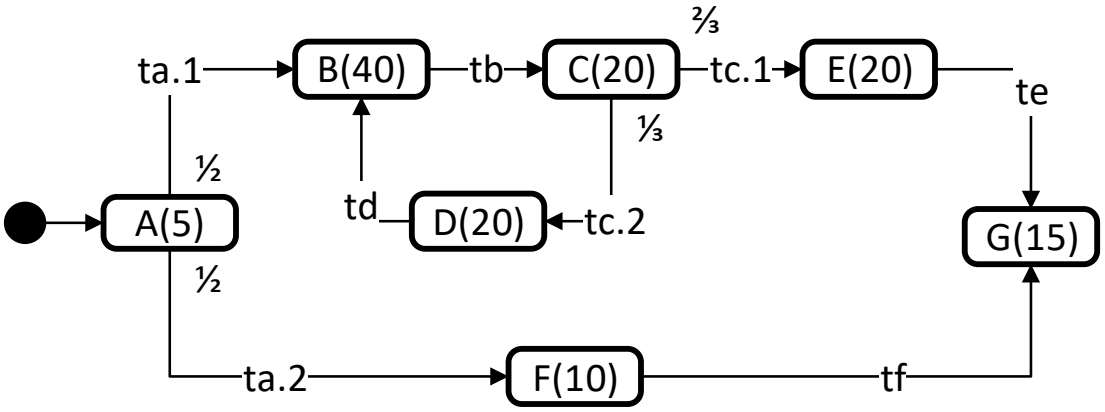

Abbildung 11: Absolute Transitionswahrscheinlichkeiten bei Prozess mit Schleife

SCC). Eine SZK ist eine Menge von Knoten in einem gerichteten Graphen, bei der jeder Knoten von jedem anderen erreicht werden kann [Tarjan, 1972, S. 155] (vgl. 2.3.4 Schleifenerkennung). Um die Wahrscheinlichkeitsberechnung zu ermöglichen legt das Modul für jede SZK einen Wurzelknoten fest, welche den Eintrittspunkt in eine SZK darstellt. Bei Prozessmodellen als gerichtete Graphen stellt eine SZK eine Schleife im Prozess dar. Im Prozessmodell von Abbildung 11 bilden die Zustände „B“, „C“ und „D“ eine SZK, da von jedem dieser Zustände jeder andere erreicht werden kann. Das Analysemodul wendet den Tarjan-Algorithmus an, um SZKs zu finden (vgl. 3.2.4 Implementierung der Schleifenerkennung). Dieser Ansatz hat allerdings den Nachteil, dass nur Schleifen mit einem einzigen Eintrittspunkt (Wurzelknoten) berechnet werden können. Darüber hinaus darf ein Zustand nur Teil einer einzigen SZK sein, d.h. die zu untersuchenden Prozesse dürfen keine verschachtelten Schleifen beinhalten. Die Verarbeitung von Prozessmodellen mit Schleifen, die über mehrere Pfade verlassen werden können, ist jedoch möglich.

Um die Durchlaufzeit des Prozesses in Abbildung 11 zu berechnen müssen die einzelnen Durchlaufzeiten anhand der zuvor definierten Regeln gewichtet und aggregiert werden. Beginnen wir mit dem Zustand „A“. Dieser ist ein direkter Nachfolger des Initialzustands des Prozesses und daher wird seine Durchlaufzeit mit der Gewichtung 1 in das Ergebnis übernommen. Der nächste Zustand ist „B“, der Wurzelknoten der SZK. Die Wahrscheinlichkeit, mit der die Schleife betreten wird (`r`) beträgt ein Drittel, daraus ergibt sich für den *rework*-Block entsprechend der zuvor definierten Formel `1-(1/r)` eine Wahrscheinlichkeit von 1,5. Dabei handelt es sich allerdings nur um die relative Wahrscheinlichkeit, da die vorgehende Transition „ta.1“ noch nicht berücksichtigt wurde. Die korrekt gewichtete Durchlaufzeit des Zustands „B“ beträgt 30 (= 40*1,5*0,5) Zeiteinheiten. Der Zustand „C“ ist mit der gleichen Wahrscheinlichkeit zu gewichten, da sich zwischen ihm

und „B“ keine Abzweigung befindet, daher wird er mit einer Durchlaufzeit von 15 (= 20*1,5*0,5) Zeiteinheiten ins Endergebnis übernommen. Der Zustand „D“ befindet sich nach einer Verzweigung und die eingehende Transition hat eine (relative) Wahrscheinlichkeit von einem Drittel. Seine Durchlaufzeit muss daher mit dieser und den vorgehenden Wahrscheinlichkeiten gewichtet werden, das inkludiert die des *rework*-Blocks und der vom Zustand „A“ ausgehenden Verzweigung. Damit ergibt sich für ihn die Dauer von 5 (= 20*(1/3)*1,5*0,5) Zeiteinheiten. Die Transition zum Zustand „E“ hat eine Wahrscheinlichkeit von zwei Dritteln. Analog zu „D“ müssen erneut die vorhergehenden Wahrscheinlichkeiten miteinbezogen werden, d.h. die des *rework*-Blocks und der Transition „ta.1“. Die Durchlaufzeit dieses Zustands beträgt demnach 10 (= 20*(2/3)*1,5*0,5). Der Zustand „F“ wird mit einer Dauer von 5 (= 10*0,5) eingerechnet. Zum letzten Zustand, „G“, sind zwei eingehenden Transitionen zu berücksichtigen. Die absolute Wahrscheinlichkeit der Transition „te“ ist mit 50% gleich wie die von „tf“, daraus resultiert eine absolute Wahrscheinlichkeit für den Zustand „G“ von 100% und seine Durchlaufzeit wird mit dem Faktor 1 gewichtet ins Ergebnis übernommen. Die gesamte Durchlaufzeit des Prozesses beträgt 85 (= 5 + 30 + 15 + 5 + 10 + 5 + 15) Zeiteinheiten.

Die vom Analysemodul angebotenen Funktionen der quantitativen Analyse werden nach ihrer Verwendung in die zwei Kategorien Ist- und What-If-Analyse unterteilt. Funktionen der erstgenannten Kategorie ermöglichen die Auswertung von Prozessen anhand der Informationen, die in den Event- bzw. State-Logs vorhanden sind. Dabei werden diese Daten analysiert ohne sie zu verändern. Die Funktionen der What-If-Analyse verwenden die gleichen Daten, allerdings geben sie den Benutzerinnen und Benutzern die Möglichkeit, sowohl Durchlaufzeiten als auch Transitionswahrscheinlichkeiten zu verändern. Dadurch kann beobachtet werden, wie sich Prozesse unter geänderten Bedingungen verhalten könnten.

2.3.2 Qualitative Analyse

Die qualitative Analyse hat das Ziel, Ansätze zur Verbesserung eines Prozesses zu finden um diesen möglichst „gut“ zu gestalten [Dumas et al., 2013, S. 185]. Dazu gibt es eine Vielzahl an Methoden mit denen Analystinnen und Analysten versuchen, Probleme in Prozessen zu identifizieren. Die Erkenntnisse aus der qualitativen Analyse werden als Input für die Optimierung bzw. dem Re-Design von Prozessen verwendet.

Die qualitative Analyse geht von der Annahme aus, dass jeder Prozess weiter verbessert werden kann. So können beispielsweise in einem bereits optimierten Prozess weiterhin Fehler vorhanden oder Missverständnisse bei der Modellierung aufgetreten sein [Hammer, 2010, S. 12]. Diese Probleme müssen gefunden und dokumentiert werden, um deren Beseitigung und eine daraus resultierende Steigerung der Leistungsfähigkeit zu ermöglichen. Des Weiteren sind auch die Beziehungen

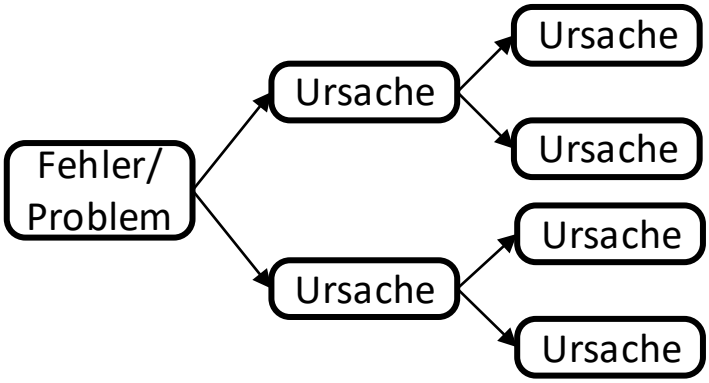

Abbildung 12: Baumdiagramm der Fehler-Ursache-Analyse (nach [Dumas et al., 2013, S. 196 f.])

zwischen den Problemen eines Prozesses von Interesse, da so deren grundlegende Ursachen erkannt werden können. Diese werden in Kausal- und Zusatzfaktoren unterschieden [Dumas et al., 2013, S. 190 ff.]. Die Eliminierung von Kausalfaktoren führt dazu, dass das daraus folgende Problem nicht mehr auftritt während eine Beseitigung von Zusatzfaktoren lediglich das Risiko dafür mindert. Der Übergang zwischen diesen Faktoren ist in der Praxis fließend, da eine saubere Abgrenzung zum einen schwierig und zum anderen oftmals nicht relevant ist. Nachfolgend wird auf ihre Unterscheidung verzichtet und stattdessen die allgemeine Bezeichnung „Faktor“ verwendet.

Eine Methode der qualitativen Analyse ist die Fehler-Ursachen-Analyse (engl. *root cause analysis*) [Dumas et al., 2013, S. 196 f.]. Sie ermöglicht das Auffinden von Schwachstellen im Prozess, d.h. Aktivitäten, die sich negativ auf die Leistungsfähigkeit des Prozesses auswirken. Das vorliegende Analysemodul bietet entsprechende Funktionen an, die Analystinnen und Analysten erlauben, Fehler-Ursachen-Analysen an MBAs durchzuführen. Zum Zwecke der qualitativen Analyse können eine Vielzahl von Informationsquellen verwendet werden, wie z.B. Befragungen der am Prozess beteiligten Personen oder Dokumentationen der Abläufe [Dumas et al., 2013, S. 196 f.]. Um MBAs zu analysieren eignen sich die Event- bzw. State-Logs, da aus diesen Informationen ihre Lebenszyklen rekonstruiert werden können.

Das Analysemodul führt eine Fehler-Ursachen-Analyse unter Verwendung eines Baumdiagramms (auch *Why-Why*-Diagramm) [Dumas et al., 2013, S. 196 f.] durch. Diese Technik ermöglicht die verkettete Darstellung von Problemen mit ihren Ursachen. Abbildung 12 bildet die Funktionsweise der Technik ab. Zu Beginn werden alle Probleme im Prozess identifiziert. Anschließend werden die zu den Problemen führenden Ursachen separat ausgewertet, dazu wird für jedes eine eigene Baumstruktur angelegt, wobei das Problem den Wurzelknoten darstellt. Durch

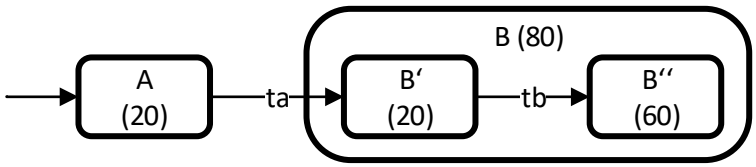

Abbildung 13: Ursachen von Problemzustand: Subzustand

die wiederholte Frage nach der Ursache („Wie kommt dieses Problem zustande?“) lässt sich der Baum immer weiter vervollständigen. Am Ende der Analyse stellen die Blattknoten des Baumes die fundamentalen Ursachen (*root causes*, [Tomić und Spasojević Brkić, 2011]) der Probleme des Prozesses dar, da sie auf keine weiteren Ursachen zurückzuführen sind.

Um problematische Zustände zu ermitteln, verwendet das Analysemodul die Durchlaufzeit von Zuständen und Prozessen. Ein Zustand wird dann als Problem angesehen, wenn seine Durchlaufzeit einen bestimmten Anteil der gesamten Durchlaufzeit des Prozesses übersteigt. Die Auswertung bezüglich der Ursachen kann in zwei Teilbereiche gegliedert werden. Erstens ist es möglich, dass die Durchlaufzeit eines zusammengesetzten Zustands aufgrund von einem oder mehrerer seiner Subzustände zu hoch ist, da sich seine Durchlaufzeit aus der Summe der Durchlaufzeiten aller Subzustände ergibt. Abbildung 13 illustriert diesen Fall mit Hilfe eines Prozessmodells. Angenommen, bei der Analyse wurde festgelegt, dass ein Zustand dann problematisch ist wenn seine Durchlaufzeit mehr als die Hälfte der Gesamtzeit beträgt. Nachdem die gesamte Durchlaufzeit 100 Zeiteinheiten beträgt, stellen die Zustände „B“ und „B”“ Probleme dar, da ihre Durchlaufzeiten mit 80 bzw. 60 Zeiteinheiten größer als der erlaubte Grenzwert von 50 (= 100*0,5) sind. Die lange Dauer von „B“ ist allerdings eine Folge der überdurchschnittlich langen Durchlaufzeit seines Subzustandes „B”“, somit ist dieser die Ursache des Problems im Zustand „B“. Für den Problemzustand „B”“ können anhand der gegebenen Informationen keine weiteren Ursachen ermittelt werden, somit stellt dieser Zustand eine fundamentale Ursache dar.

Um die Ursachen problematischer Zustände zu finden wertet das Analysemodul neben den Durchlaufzeiten von Subzuständen problematischer Zustände auch vertikale Synchronisationsabhängigkeiten (vgl. 2.1.2 Synchronisation von mehrstufigen Prozessmodellen) aus. Vorbedingungen in Transitionen können verhindern, dass ein Zustand verlassen wird, beispielsweise wenn darauf gewartet werden muss, dass sich die Prozessmodelle aller Konkretisierungen des MBAs einer Ebene in einem festgelegten Zustand befinden. Das führt zu einer erhöhten Durchlaufzeit im wartenden Zustand, wodurch dieser ein Problem im Prozess darstellen kann. Das Analysemodul untersucht die in Abschnitt 2.2.4 beschriebenen Prädikate. Entscheidend ist dabei, mit Hilfe der Event- und State-Logs festzustellen ob der be-

troffene Problemzustand kurz nach dem Erfüllen der Vorbedingung einer seiner Transitionen verlassen wurde. Falls das zutrifft ist das Problem auf den Prozess der in der Synchronisationsbeziehung angegebenen Ebene zurückzuführen. Beim Analysieren der Event- und State-Logs wird davon ausgegangen, dass die Prozesse der MBAs korrekt ausgeführt wurden, d.h. dass entsprechende Vorbedingungen beim Auslösen einer Transition erfüllt waren.

Abbildung 4 zeigt ein MBA mit zwei Ebenen sowie einer Synchronisationsabhängigkeit. Wie zuvor wird angenommen, dass ein Zustand dann problematisch ist, wenn seine Durchlaufzeit mehr als die Hälfte der Gesamtzeit ausmacht. Diese beträgt bei dem Prozessmodell der Ebene „l1“ 100 Zeiteinheiten, demnach ist der Zustand „A“ mit einer Dauer von 70 ein Problem. Dieser kann nur über die Transition „ta“ verlassen werden, welche die Vorbedingung besitzt dass sich alle Konkretisierungen der Ebene „l2“ im Zustand „D“ befinden müssen (`cond="$_everyDescendantAtLevelIsInState('l2', 'D')"`, vgl. 2.2.4 Prädikate für Synchronisationsabhängigkeiten). Wird der Zustand „A“ verlassen, kurz nachdem alle Konkretisierungen der Ebene „l2“ „D“ betreten haben, kann davon ausgegangen werden, dass „A“ bereits fertig abgearbeitet war und lediglich auf die Erfüllung der Vorbedingung der Transition wartete. In diesem Fall ist die Ursache für das Problem von „A“ im Prozess der Ebene „l2“ zu suchen, beginnend mit dem im Prädikat angegebenen Zustand „D“. Dieser ist nicht problematisch, da seine Durchlaufzeit weniger als die Hälfte der Gesamtzeit beträgt ($20 < (60 + 20)*0{,}5$) und so müssen seine Vorgänger betrachtet werden. Der Zustand „C“ dauert länger als die Hälfte der gesamten Durchlaufzeit und stellt damit ein Problem dar. Daraus folgt dass „C“ eine Ursache für das Problem von „A“ ist, da durch ihn die Erreichung des in der blockierenden Synchronisationsabhängigkeit spezifizierten Zustands „D“ und somit das Verlassen von „A“ verzögert wird. Würde der Prozess auf der Ebene „l2“ keine problematischen Zustände enthalten wäre das Problem auf den gesamten Prozess zurückzuführen.

Um feststellen zu können, ob der Zeitpunkt des Verlassens eines Zustands kurz nach dem Erfüllen der Transitionsbedingung erfolgte, muss zwischen den unterstützten Prädikaten unterschieden werden. Eine Vorbedingung bestehend aus dem Prädikat `$_everyDescendantAtLevelIsInState('l2', 'D')` ist dann erfüllt, wenn sich alle Konkretisierungen auf der angegebenen Ebene („l2“) im entsprechenden Zustand („D“) befinden. Dies passiert zu dem Zeitpunkt, in dem die letzte Konkretisierung der Ebene in diesen Zustand wechselt. Unter der Annahme, dass die Prozessmodelle korrekt abgearbeitet wurden, werden bei der nachträglichen Auswertung durch das Analysemodul nur jene Konkretisierungen beachtet, die sich zum Zeitpunkt des Betretens der Transition mit der Vorbedingung in dem entsprechenden Zustand befanden, da nach dem Feuern der Transition neue Konkretisierungen erzeugt worden sein könnten. Um dieses Vorgehen zu verdeutlichen

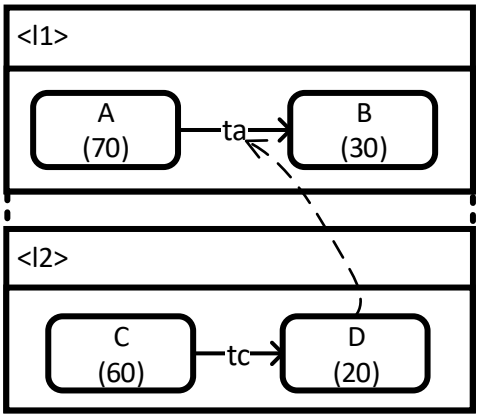

Abbildung 14: Ursachen von Problemzustand: Synchronisationsabhängigkeiten

sind in Quellcode 14 State-Logs der zum Zeitpunkt der Analyse existierenden Konkretisierungen des MBAs von Abbildung 14 abgebildet. Das State-Log des MBAs selbst ist in Quellcode 13 zu sehen, demnach wurde der Zustand „A“ am 1.1.2016 um 11:02 verlassen. Zu diesem Zeitpunkt befanden sich zwei der drei Konkretisierungen in dem entsprechenden Zustand „D“, die erste betrat ihn um 11:00 Uhr und die zweite um 10:00 Uhr. Bei der dritten Konkretisierung wurde „D“ erst um 16:00 aktiv, daher kann diese erst erzeugt worden sein, nachdem die Transition „ta“ gefeuert wurde. Hätte sie bereits vorher existiert, wäre das Prozessmodell vom SCXML-Interpreter falsch ausgeführt worden. Da von einer korrekten Ausführung ausgegangen wird, war die Vorbedingung des Zustandsübergangs „ta“ um 11:00 erfüllt, d.h. zu dem Zeitpunkt als die letzte Konkretisierung in den Zustand „D“ wechselte. Zwei Minuten später wurde „A“ verlassen, daher ist die Wahrscheinlichkeit hoch, dass er durch die Synchronisationsabhängigkeit blockiert wurde.

Quellcode 13: State-Logs des MBAs aus Abbildung 14

```
1 <stateLog>
2   <state ref="A"
3     from="2016-01-01T08:00:00.000+02:00"
4     until="2016-01-01T11:02:00.000+02:00" />
5   ...
6 </stateLog>
```

Quellcode 14: State-Logs von Konkretisierungen des MBAs von Abbildung 14 der Ebene „l2“

```
1 <stateLog>
2   ...
3   <state ref="D"
4     from="2016-01-01T11:00:00.000+02:00" />
5 </stateLog>
```

```
6  <stateLog>
7    ...
8    <state ref="D"
9      from="2016-01-01T10:00:00.000+02:00" />
10 </stateLog>
11 <stateLog>
12   ...
13   <state ref="D"
14     from="2016-01-01T16:00:00.000+02:00" />
15 </stateLog>
```

Das Prädikat `$_someDescendantAtLevelIsInState` wird analog zu `$_everyDescendantAtLevelIsInState` ausgewertet. Eine Transition mit dieser Vorbedingung kann dann genommen werden, wenn sich zumindest eine Konkretisierung in dem angegebenen Zustand befindet. Dadurch wird die Vorbedingung zu dem Zeitpunkt erfüllt, in dem die erste Konkretisierung in den spezifizierten Zustand wechselt. Hierbei werden vom Analysemodul erneut nur jene Konkretisierungen beachtet, die sich zum Zeitpunkt, in dem der problematische Zustand verlassen wurde, in dem spezifizierten Zustand befunden haben. Konkretisierungen, die sich zuvor oder danach in diesem Zustand befanden werden nicht miteinbezogen, da sie nicht zum Auslösen der Transition beigetragen haben.

Die drei übrigen Prädikate `$_isDescendantAtLevelInState`, `$_isAncestorAtLevelInState` und `$_ancestorAtLevelIsInState` beziehen sich bei ausschließlicher Verwendung von einfachen Konkretisierungshierarchien jeweils auf einzelne Konkretisierungen bzw. Vorgänger. Daher muss das Analysemodul in diesen Fällen beim Ermitteln des Zeitpunkts, zu dem die entsprechenden Vorbedingungen erfüllt wurden, keine weiteren Einschränkungen bei den zu beachtenden MBAs vornehmen. Die Erfüllung findet jedes Mal statt, wenn der spezifizierte Zustand betreten wird.

Auf Basis der durch die Fehler-Ursachen-Analyse gewonnenen Erkenntnisse können Analystinnen und Analysten Prozesse zielgerichteter untersuchen. Verbesserungsmaßnahmen sollten zuerst an den fundamentalen Ursachen angesetzt werden, allerdings können sie unter Umständen, z.B. wenn sie außerhalb des Geltungsbereichs eines Unternehmens liegen, nicht beseitigt werden [Dumas et al., 2013, S. 185 ff.].

2.3.3 Analyse von Multilevel Business Artifacts

Dieser Abschnitt beschreibt, wie MBAs hinsichtlich ihrer Durchlaufzeiten (siehe auch [zur Mühlen und Shapiro, 2010, S. 7]) ausgewertet werden können. Darüber hinaus wird darauf eingegangen, wie durch die Navigation zwischen MBAs und ihren Konkretisierungen weitere Informationen gewonnen werden können.

Da ein MBA eine Instanz seines Top-Levels ist, handelt es sich bei Maßzahlen auf dieser Ebenen immer um konkrete Werte [Schuetz, 2015, S. 197 ff.]. In den darunterliegenden Ebenen werden hingegen die aggregierten Daten der jeweiligen Konkretisierungen dargestellt. Das Analysemodul verwendet dabei den arithmetischen Durchschnitt als Aggregatsfunktion. Die durchschnittliche Durchlaufzeit eines Zustands wird berechnet, indem die tatsächlichen Durchlaufzeiten aus den Konkretisierungen aufsummiert und die Summe durch deren Anzahl dividiert wird.

Abbildung 15 zeigt ein MBA mit den Durchlaufzeiten der Zustände, welche jeweils unter der Bezeichnung in Klammern vermerkt sind. In dem Beispiel befindet sich das MBA „MyInsuranceCompany“ bereits seit 250 Zeiteinheiten im Zustand „Working“. Der Zustand „Closed“ wurde noch nicht betreten, daher ist dort auch noch keine Dauer angegeben. Des Weiteren besitzt das MBA „MyInsuranceCompany“ unter dem Top-Level die Ebenen „tacticalInsurance“ und „operationalInsurance“. Die Zeitangaben der Zustände auf diesen Ebenen sind aggregierte Werte, das bedeutet beispielsweise dass sich die Konkretisierungen des MBAs „MyInsuranceCompany“ auf der Ebene „tacticalInsurance“ im Durchschnitt 25 Zeiteinheiten im Zustand „ChooseProducts“ befinden. Falls es sich beim Zustand „Finished“ um einen *Archival State* handelt und dieser daher von der Berechnung der gesamten Durchlaufzeit ausgeschlossen wird, beträgt diese nach den Regeln der *Flow Analysis* (vgl. 2.3.1 Quantitative Analyse) für die Ebenen „tacticalInsurance“ 125 (= 25 + 100) Zeiteinheiten.

Damit die Daten korrekt interpretiert werden können, muss die Art der Berechnung bekannt sein. So gibt es verschiedene Ansätze, wie die Grundmenge eingeschränkt werden kann (siehe auch [Schuetz, 2015, S. 200]). Zum Beispiel können zur Berechnung der durchschnittlichen Durchlaufzeit nur jene Konkretisierungen herangezogen werden, bei denen der gesuchte Zustand zumindest einmal betreten wurde. Eine andere Variante wäre nur diejenigen Konkretisierungen zu betrachten, bei denen der gesuchte Zustand aktiv war und die sich zur Zeitpunkt der Analyse in einem bestimmten Zustand befinden. Die erste Variante hat den Vorteil, dass Daten schneller für die Analyse verfügbar sind. Das kommt daher, dass schon mit dem Verlassen des gesuchten Zustands seine Durchlaufzeit für die Kalkulation verwendet werden kann und im Vergleich zur zweiten Variante nicht erst darauf gewartet werden muss, bis ein bestimmter Zustand aktiv ist. Dadurch sind die zur Berechnung des Durchschnitts benötigten Daten früher verfügbar und können schneller ausgewertet werden. Die Aussagekraft dieser Daten ist allerdings nur bedingt gegeben, da die Grundmenge an Daten bei Zuständen, die im Prozess früher auftreten größer ist als bei späteren Zuständen. Dadurch sind Aussagen über die Durchlaufzeit früherer Zustände robuster gegenüber Ausreißern und, verglichen mit späteren Zuständen, tendenziell genauer. Bei der zweiten Berechnungsvariante tritt dieses Problem nicht auf, da sich alle betrachteten MBAs im gleichen

Zustand befinden und die Ergebnisse somit die gleiche Aussagekraft besitzen. Hier sind jedoch die Daten erst später verfügbar und so kann es passieren, dass Trends nicht zeitnah erkannt werden können.

Um den Detaillierungsgrad von den Ergebnissen der quantitativen Analyse zu verändern, können Analystinnen und Analysten entlang der Konkretisierungshierarchie navigieren [Schuetz, 2015, S. 203 f.]. Die Navigation von einem MBA zu einem seiner Konkretisierungen ist äquivalent zu einer *slice*-Operation [Farkisch, 2011, S. 39] in einem Data-Warehouse, weil dadurch die Wertebasis für die Berechnung der Durchlaufzeit eingeschränkt wird. Daraus resultieren konkretere Werte bei der Berechnung von Durchlaufzeiten. Des Weiteren entspricht diese Navigation einer *drill-down*-Operation [Farkisch, 2011, S. 39], da anstatt einer aggregierten Durchlaufzeit für den Prozess einer Ebene ein Ergebnis pro Konkretisierung vorliegt. Umgekehrt verhält sich eine Navigation von einer Konkretisierung zu ihrem übergeordnetem MBA: die Wertebasis wird vergrößert und die resultierenden Durchlaufzeiten werden allgemeiner.

Abbildung 15 illustriert, wie durch *slice/drill-down*-Operationen der Detaillierungsgrad von Analysen erhöht werden kann. Die Analyse der Ebene „tacticalInsurance“ des MBAs „MyInsuranceCompany“ liefert aggregierte Werte, d.h. durchschnittliche Durchlaufzeiten aller Konkretisierungen. Über die Navigation entlang der Konkretisierungshierarchie zu Konkretisierungen dieser Ebene erhalten Analystinnen und Analysten die tatsächlichen Durchlaufzeiten für jede Instanz anstatt eines zusammengefassten Durchschnitts. Man beachte die unterschiedlichen Durchlaufzeiten auf der Ebene „operationalInsurance“. Durch die Navigation wird die Grundmenge für die Berechnung eingeschränkt, da das MBA „MyCarInsurance“ nur eigene Konkretisierungen berücksichtigt.

Wurde das Prozessmodell eines MBAs von einer Konkretisierung spezialisiert, können daraus weitere Informationen gewonnen werden. Die Konkretisierung „MyCarInsurance“ verfeinert den Zustand „DevelopProduct“, indem zwei Subzustände hinzugefügt wurden. Somit kann seine Durchlaufzeit durch eine Navigation von „MyInsuranceCompany“ zu „MyCarInsurance“ mithilfe einer *drill-down*-Operation innerhalb des Zustands genauer aufgeschlüsselt werden [Schuetz, 2015, S. 207]. Die (tatsächliche) Durchlaufzeit von „DevelopProduct“ beträgt 95 Zeiteinheiten und setzt sich aus den Subzuständen „CheckFeasiblity“ (15) und „ImplementProduct“ (80) zusammen. Wie in Abschnitt 2.1.1 beschrieben, können Transitionen ebenfalls verfeinert werden. Im Beispiel wird das Ereignis der Transition „finish“ auf der Ebene „tacticalInsurance“ von „MyInsuranceCompany“ in der Konkretisierung „MyCarInsurance“ zu „finish.abort“ und „finish.success“. Entsprechend der definierten Regeln enthält die Konkretisierung nicht mehr die ursprüngliche Transition. Da diese beiden Spezialisierungen durch das ursprüngliche Ereignis „finish“ ausgelöst werden [Barnett et al., 2015, #events], müssen sie bei

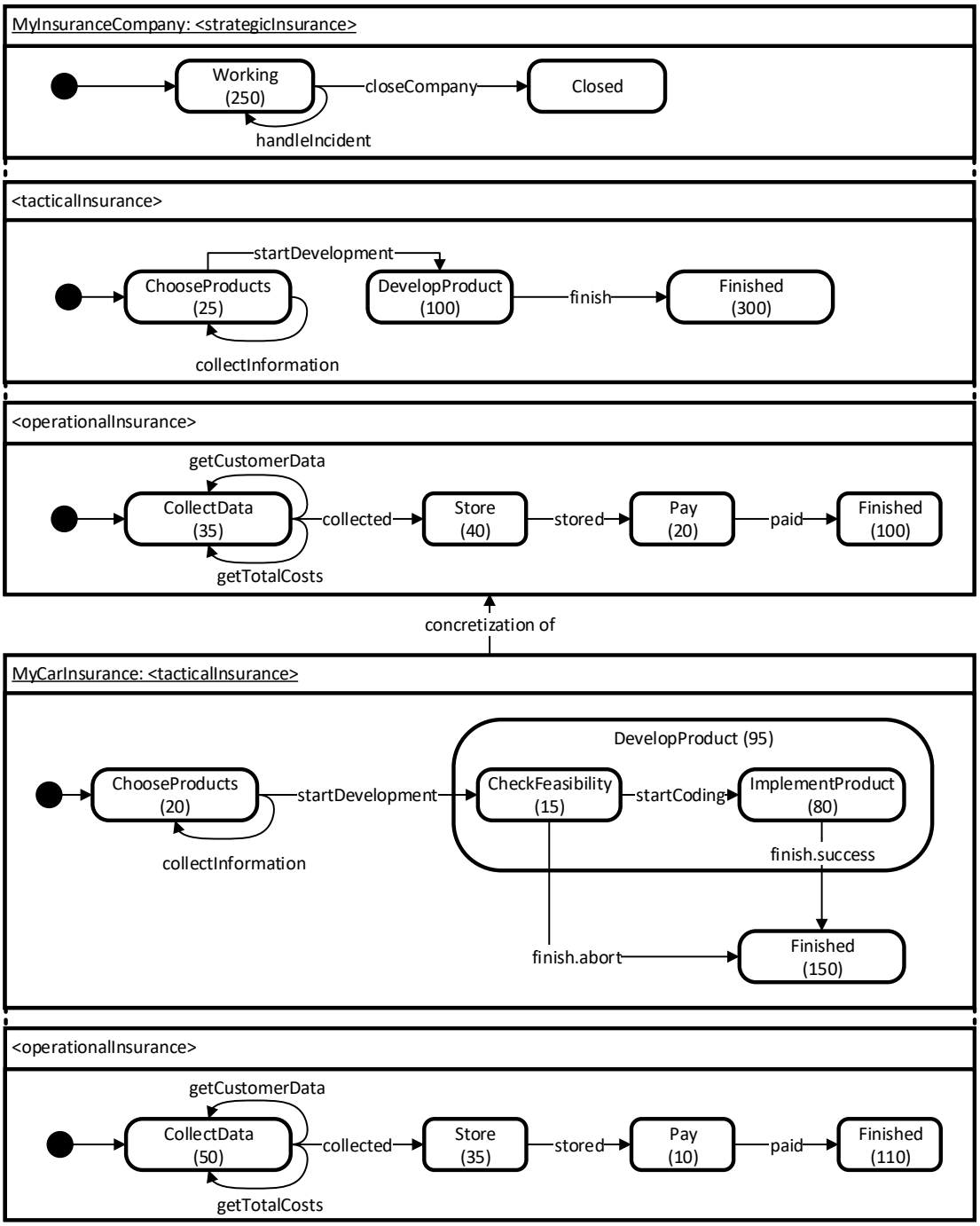

Abbildung 15: Durchlaufzeiten einer hetero-homogenen Konkretisierungshierarchie

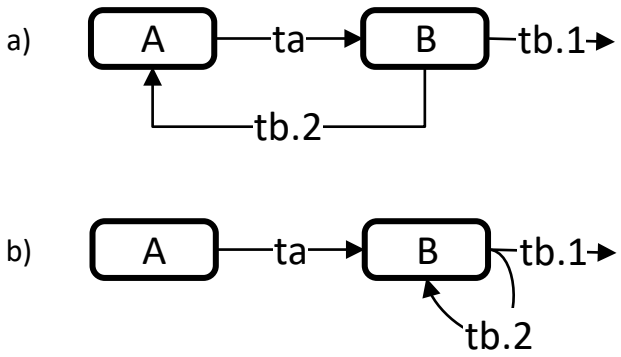

Abbildung 16: Schleifen

der Berechnung der Transitionswahrscheinlichkeit der allgemeineren Transition im MBA „MyInsuranceCompany“ miteinbezogen werden. In den Konkretisierungen werden die Wahrscheinlichkeiten allerdings für jede Transition separat berechnet.

2.3.4 Schleifenerkennung

In Prozessen ist es möglich, dass bestimmte Abschnitte mehrmals durchlaufen werden. Ein Beispiel hierfür wäre die Qualitätskontrolle nach einem Herstellungsprozess: liegt die Qualität des Ausgangsprodukts unter einem festgelegten Grenzwert, ist eine Nachbearbeitung des Produkts erforderlich und der Prozess oder Teile davon müssen wiederholt werden.

Die Prozessmodelle in MBAs entsprechen gerichteten Graphen mit Knoten (Zustände) und Kanten (Zustandsübergänge) [Domschke und Drexl, 2005, S. 65 ff.]. Ein Pfad in einem Graph ist eine Folge von Kanten, die von einem Knoten zu einem anderen führen. Schleifen sind ein Spezialfall von Pfaden, da bei ihnen Ausgangs- und Endknoten identisch sind [Tarjan, 1972, S. 146]. Das Prozessmodell a) in Abbildung 16 beinhaltet einen Pfad, der ausgehend vom Zustand „A“ über die Transitionen „ta“ und „tb.2“ wieder zurück zum Ausgangszustand führt und somit einer Schleife entspricht. In dieser Arbeit sind Pfade, die nicht über zumindest einen anderen Zustand verlaufen, von der Schleifendefinition ausgenommen. Ein Beispiel hierfür ist die Transition „tb.2“ im Prozessmodell b) in Abbildung 16. Da solche Transitionen nicht zu einer Änderung des aktiven Zustands führen, sind diese nicht relevant für die Berechnung von Durchlaufzeiten und werden nicht in die Auswertung der Prozessmodelle miteinbezogen.

Eine Menge von Knoten in einem gerichteten Graphen, bei denen es von jedem Knoten einen Pfad zu allen anderen gibt, wird als stark zusammenhängende Komponente (SZK) bezeichnet [Walther, 1978, S. 11]. Im Modell a) von Abbildung 16 bilden die Knoten „A“ und „B“ zusammen eine SZK. Mit diesem Konzept können

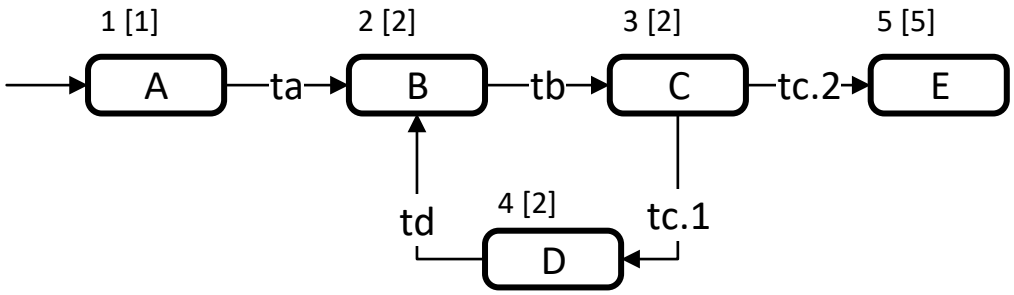

Abbildung 17: Index und *Lowlinks* von SZKs

Schleifen in Prozessmodellen gefunden werden, das Analysemodul verwendet dazu den Algorithmus von Tarjan zur Erkennung von SZKs. Dieser führt eine Tiefensuche (*depth-first search*) auf einem gerichteten Graphen durch, womit jeder Knoten genau einmal besucht wird [Turau, 2009, S. 90 ff.]. Der Algorithmus legt die Knoten in der Reihenfolge, in der sie besucht werden, auf einem Stack ab und weist ihnen einen Index zur Identifizierung sowie eine *Lowlink* Variable zu. Der *Lowlink* eines Knotens ist der kleinste Index aller Knoten, die ausgehend von ihm erreicht werden können. Ist der Index eines Knotens gleich seines *Lowlinks* handelt es sich um die Wurzel einer SZK, die anschließend gebildet wird. Dazu werden Knoten vom Stack geholt und der aktuellen SZK hinzugefügt. Wurde der Wurzelknoten der SZK gefunden, ist diese komplett und der Algorithmus fährt fort, indem er den nächsten zu verarbeitenden Knoten besucht. [Tarjan, 1972, S. 156] Die Implementierung dieses Algorithmus in XQuery ist in Abschnitt 3.2.4 beschrieben.

Abbildung 17 zeigt einen gerichteten Graphen mit den jeweiligen Indizes der Knoten und den *Lowlinks* in Klammern. Bei den Zuständen „A“ bzw. „E“ sind Index und *Lowlink* jeweils gleich, daher bilden sie eigenständige SZKs. Die Zustände „B“, „C“ und „D“ verfügen über den gleichen *Lowlink*, was bedeutet, dass sie sich in derselben SZK befinden. Von jedem dieser Zustände kann demnach der Zustand mit dem Index 2, d.h. „B“, erreicht werden. Dieser Zustand ist die Wurzel der SZK.

Durch die Verwendung dieses Ansatzes ergeben sich in Kombination mit der Wahrscheinlichkeitsberechnung von Schleifen (vgl. 2.3.1 Quantitative Analyse) folgende Einschränkungen, denen die zu berechnenden Modelle unterworfen sind. Das Analysemodul erlaubt die Berechnung von Schleifen, wenn diese nur über einen Zustand betreten werden können. Daraus folgt, dass eine SZK immer genau einen eindeutigen Wurzelknoten besitzen muss. Des Weiteren werden verschachtelte Schleifen nicht erkannt, zu sehen in Abbildung 18. Dieses Prozessmodell besitzt die aus den Zuständen „A“, „B“, „C“ und „D“ bestehende SZK, die jedoch eine

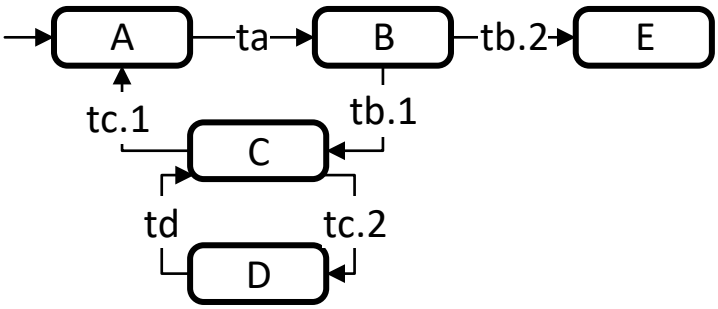

Abbildung 18: Verschachtelte Schleife

weitere Schleife aus den Zuständen „C“ und „D“ besitzt. Diese kann allerdings nicht durch den Algorithmus von Tarjan erkannt werden.

3 Quantitative Analyse

Dieses Kapitel beschreibt jene Funktionen des Analysemoduls, welche die quantitative Auswertung von MBAs ermöglichen. Zuerst wird erläutert, wie das Modul von Benutzerinnen und Benutzern verwendet werden kann. Im Anschluss daran folgt eine Darstellung der Umsetzung in XQuery mitsamt den wichtigsten Hilfsfunktionen.

3.1 Verwendung

Die Funktionen der quantitativen Analyse des Moduls werden in Ist- und What-If-Analyse unterteilt. Erstere berechnen die Durchlaufzeiten eines MBAs mithilfe der unveränderten Daten aus Event-Logs, während letztere die Ermittlung von Durchlaufzeiten unter veränderten Bedingungen ermöglichen. Die folgenden Abschnitte gehen genauer auf diese beiden Kategorien ein.

3.1.1 Ist-Analyse

Dieser Abschnitt beschreibt jene Funktionen des Analysemoduls, die den Ist-Zustand von MBAs abbilden. Ihre Schnittstellen sind in Tabelle 4 zusammengefasst. Auf Basis von Event-Logs werden Metriken errechnet, durch die Aussagen über den Prozess getroffen werden können. Die Rückgabewerte der nachstehenden Funktionen sind jeweils vom XML-Schema Datentyp `xs:duration`.

Die Funktion `getCycleTimeOfInstance` liefert die Durchlaufzeit vom Prozessmodell des Top-Levels eines MBAs. Ihr müssen zwei Parameter übergeben werden. Der erste, `$mba`, ist das MBA selbst als XML-Element und der zweite, `$toState`, beschreibt den Zustand, bis welchem die Zeit berechnet werden soll. Letzterer ist optional, wird für ihn eine leere Menge übergeben gibt die Funktion die komplette Durchlaufzeit vom ersten bis zum letzten Zustand zurück. Sollte die Instanz den Zustand mit der Bezeichnung `$toState` noch nicht erreicht haben, wird eine leere Menge zurückgegeben.

Aufbauend auf `getCycleTimeOfInstance` berechnet `getTotalActualCyleTime` die durchschnittliche Durchlaufzeit aller Konkretisierungen eines MBAs auf einer bestimmten Ebene. Zusätzlich zu den beiden Parametern `$mba` und `$toState` benötigt sie noch die Bezeichnung der zu analysierenden Ebene. Bei der Berechnung der Durchlaufzeit werden nur Konkretisierungen berücksichtigt, die `$toState` bereits erreicht haben bzw. sich gerade in diesem Zustand befinden.

Die Funktion `getAverageCycleTime` wurde von [Schuetz, 2015, S. 206] übernommen und ermöglicht die Berechnung der durchschnittlichen Durchlaufzeit eines Zustands. Die Parameter `$mba` und `$level` haben hier die gleiche

Tabelle 4: Schnittstelle quantitative Analyse

Funktion	**Parameter**	**Beschreibung**
getCycleTimeOfInstance	$mba as element(), $toState as xs:string?	Liefert Durchlaufzeit des Top-Levels eines MBAs.
getTotalActualCycleTime	$mba as element(), $level as xs:string, $toState as xs:string?	Berechnet die tatsächliche Durchlaufzeit des Prozessmodells einer bestimmten Ebene eines MBAs.
getAverageCycleTime	$mba as element(), $level as xs:string, $inState as xs:string?, $stateId as xs:string	Liefert die durchschnittliche Durchlaufzeit eines Zustands.
getTotalCycleTimeInStates	$mba as element(), $states as xs:string*	Ermittelt die Zeit, die ein MBA durchschnittlich in bestimmten Zuständen verweilt.
getTotalCycleTime	$mba as element(), $level as xs:string, $inState as xs:string?, $excludeArchiveStates as xs:boolean?, $changedStates as element()*, $changedTransitions as element()*, $changedTransitionsFactors as xs:decimal*	Ermittelt die Durchlaufzeit eines MBAs unter geänderten Bedingungen.
getTotalCycleTimeToState	$mba as element(), $level as xs:string, $inState as xs:string?, $toState as xs:string, $changedStates as element()*, $changedTransitions as element()*, $changedTransitionsFactors as xs:decimal*	Berechnet die durchschnittliche Zeit, bis ein Zustand erreicht wird.

Bedeutung wie bei den vorhergehenden Funktionen. Dazu kommen `$stateId` und `$inState`. Der erstgenannte Parameter ist die Bezeichnung des gesuchten Zustands. Der zweite, `$inState`, ist optional und schränkt die Grundmenge zur Berechnung des Durchschnitts ein. Es werden nur jene Konkretisierungen verwendet, die sich zum Zeitpunkt der Analyse in dem genannten Zustand befinden. Falls kein Wert für `$inState` übergeben wird, berechnet die Funktion die Durchlaufzeit über alle Konkretisierungen des übergebenen MBAs auf der angegebenen Ebene.

Die Funktion `getTotalCycleTimeInStates` unterstützt Analystinnen und Analysten beim Ermitteln der Zeit, die in verschiedenen Zuständen verbracht wurde. Die Funktion benötigt zum einen das MBA als XML-Element und zum anderen eine Sequenz von Zustandsbezeichnungen. Ausgewertet wird der Top-Level des MBAs. Wurde ein Zustand in dieser Sequenz noch nicht erreicht, wird er nicht in das Ergebnis mit eingerechnet. Der Parameter `$states` unterliegt der Einschränkung, dass sich in dieser Sequenz keine Zustände befinden dürfen, die in einer hierarchischen Beziehung zueinander stehen. Dies würde dazu führen, dass Durchlaufzeiten mehr als einmal eingerechnet werden und das Ergebnis verfälscht wird. Das vorliegende Analysemodul führt keine Überprüfung der in der Sequenz angegeben Zustände auf hierarchische Abhängigkeiten durch, daher obliegt diese Verantwortung den Benutzerinnen und Benutzern. Ein Beispiel für die Verwendung dieser Funktion ist die Berechnung der Bearbeitungs- bzw. Wartezeit eines Prozesses (vgl. 2.3.1 Quantitative Analyse), vorausgesetzt das Prozessmodell wurde um die dafür benötigten Bearbeitungs- und Wartezustände erweitert. Um die Effizienz des Prozesses zu berechnen, muss die Bearbeitungszeit durch die gesamte Durchlaufzeit dividiert werden [Dumas et al., 2013, S. 224]. Die Bearbeitungszeit eines MBAs kann mit Hilfe von `getTotalCycleTimeInStates` bestimmt werden, die gesamte Durchlaufzeit liefert `getCycleTimeOfInstance`.

3.1.2 What-If-Analyse

Neben Funktionen zur Darstellung des Ist-Zustands stellt das Analysemodul Funktionalität zur Durchführung von What-If-Analysen bereit. Durch Variationen der vorliegenden Daten kann überprüft werden, wie Prozessmodelle auf Veränderungen der Durchlaufzeiten von Zuständen bzw. Transitionswahrscheinlichkeiten reagieren.

Die Funktion `getTotalCycleTime` berechnet die durchschnittliche Durchlaufzeit von Konkretisierungen eines MBAs auf einer bestimmten Ebene, wobei wie bei `getAverageCycleTime` nur Instanzen berücksichtigt werden die sich im angegebenen Zustand `$inState` befinden. Die Funktion ermöglicht das Weglassen von sogenannten *Archival States*. Diese können anstelle von Finalzuständen (`final`-Elementen) verwendet werden und verhindern, dass ein Objekt beim Ende

seiner Lebensdauer gelöscht wird [Schuetz, 2015, S. 62]. *Archival States* müssen allerdings vor der Analyse mithilfe des Attributs `mba:isArchiveState` als solche gekennzeichnet werden. Das Attribut muss zu den jeweiligen Zuständen hinzugefügt und auf „true“ gesetzt werden, ein Beispiel hierfür ist in Quellcode 15 zu sehen. Im Prozessmodell eines MBAs können mehrere Zustände als *Archival State* definiert werden.

Quellcode 15: Kennzeichnung von *Archival States*

```
1 <sc:state id="A" mba:isArchiveState="true">
2   ...
3 </sc:state>
```

Zur Veränderung der vorliegenden Durchlaufzeiten und Transitionswahrscheinlichkeiten werden die optionalen Parameter `$changedStates`, `$changedTransitions` und `$changedTransitionsFactors` verwendet. Quellcode 16 zeigt das Format von `$changedStates`. Er besitzt ein Wurzelelement `states`, unter dem sich eine Sequenz von `state`-Elementen befindet. Dabei handelt es sich um leere Knoten mit den beiden Attributen `id` sowie `factor`. Für die What-If-Analyse berechnet sich das Modul die durchschnittliche Zeit der angegebenen Zustände und multipliziert sie, falls vorhanden, mit dem entsprechenden Faktor. In dem Beispiel von Quellcode 16 würde die Durchlaufzeit des Zustands „A“ verdreifacht und die von „B“ halbiert werden.

Quellcode 16: Format des Parameters `$changedStates`

```
1 <states>
2   <state id="A" factor="3"/>
3   <state id="B" factor="0.5"/>
4 </states>
```

Um Transitionswahrscheinlichkeiten zu verändern werden die zwei Parameter `$changedTransitions` und `$changedTransitionsFactors` benötigt. Ersterer beinhaltet die Transitionen, deren Wahrscheinlichkeit verändert werden soll, letzterer die zu verwendenden Wahrscheinlichkeiten. Der Umstand, dass zwei Parameter verwendet werden müssen, ist dem Fehlen von eindeutigen Bezeichnern der Transitionen geschuldet. Daher muss mithilfe der *Node-Identity* [W3C, 2015, #node-identity] gearbeitet werden. Das bedeutet für die Verwendung des Moduls, dass der Parameter `$changedTransitions` Transitionen enthalten muss, die aus dem zu analysierenden SCXML-Modells selektiert wurden. Der Parameter `$changedTransitionsFactors` ist eine Sequenz von Dezimalzahlen, welche die zu verwendende Wahrscheinlichkeit festlegen. Die Reihenfolge der Transitionen in `$changedTransitions` muss der Reihenfolge der Faktoren in `$changedTransitionsFactors` entsprechen. Alternativ könnte die Angabe veränderter Transitionen analog zu der veränderter Zustände erfolgen. In diesem Fall

müsste eine Liste mit `transition`-Elementen mit vier identifizierenden Attributen (Bezeichnung von Ausgangs-/Zielzustand, Transitionsbedingung und auslösendes Ereignis) verwendet werden. Es ist anzumerken, dass das Analysemodul keine Überprüfung der vergebenen Wahrscheinlichkeiten durchführt. Das bedeutet, dass Benutzerinnen und Benutzer darauf achten müssen, dass die Summe der Wahrscheinlichkeiten von ausgehenden Transitionen eines Zustands genau 100% ergibt. Das gleiche gilt für eingehende Transitionen.

Im Unterschied zu `getTotalCycleTime` verwendet `getTotalCycleTimeToState` keinen Parameter `$excludeArchiveStates`, sondern `$toState`. Dieser funktioniert wie bei den Funktionen der Ist-Analyse. Es wird die durchschnittliche Durchlaufzeit berechnet, bis der Zustand mit der Bezeichnung `$toState` zum ersten Mal erreicht wurde.

3.2 Technische Dokumentation

In den folgenden Abschnitten wird die Umsetzung der in Abschnitt 3.1 beschriebenen Funktionen erläutert. Darunter fallen neben der Berechnung von Transitionswahrscheinlichkeiten und durchschnittlichen Durchlaufzeiten auch die Ermittlung von Transitionen, die dazu führen, dass ein bestimmter Zustand aktiv wird sowie die Schleifenerkennung.

3.2.1 Durchschnittliche Durchlaufzeit von Prozessen

Dieser Abschnitt beschreibt die Umsetzung der XQuery-Funktionen zur Berechnung der Durchlaufzeit von Prozessen. Der erste Teil widmet sich mit `getTotalActualCycleTime`, `getCycleTimeOfInstance`, `getAverageCycleTime` und `getTotalCycleTimeInStates` den Funktionen der Ist-Analyse. Der zweite Teil behandelt die beiden Funktionen für die What-If-Analyse von Prozessmodellen, `getTotalCycleTime` und `getTotalCycleTimeToState`.

Die Funktion `getTotalActualCycleTime` wird in Quellcode 17 dargestellt. Zuerst werden alle Konkretisierungen des übergebenen MBAs mithilfe von `getDescendantsAtLevelOrMBA` geholt. Diese Funktion prüft, ob es sich bei der übergebenen Ebene um den Top-Level des MBAs handelt. Wenn das zutrifft, wird das unveränderte MBA zurückgegeben. Ansonsten liefert die Funktion alle Konkretisierungen des MBAs der gesuchten Ebene als eine Sequenz von `mba`-Elementen zurück. Anschließend folgt eine Iteration über alle Elemente der zurückgelieferten MBAs, wobei die Durchlaufzeit für jedes Element von der Funktion `getCycleTimeOfInstance` (vgl. Quellcode 18) berechnet wird. Dabei ist es möglich, dass ein MBA der Sequenz `$descendants` noch nicht den Zustand `$toState` erreicht hat. In diesem Fall würde anstatt einer Zeitspanne vom Typ

Quellcode 17: Funktion getTotalActualCycleTime

```
1  declare function analysis:getTotalActualCycleTime(
2      $mba as element(),
3      $level as xs:string,
4      $toState as xs:string?
5  ) as xs:duration? {
6    let $descendants :=
7      analysis:getDescendantsAtLevelOrMBA($mba, $level, $toState)
8
9    let $cycleTimes :=
10     for $descendant in $descendants
11     return analysis:getCycleTimeOfInstance($descendant, $toState)
12
13   return fn:avg($cycleTimes)
14 };
```

`xs:duration` eine leere Menge zurückgeliefert werden. Abschließend wird mit der Built-In Funktion `avg` der Durchschnitt über die Sequenz `$cycleTimes` gebildet, wobei leere Mengen ignoriert werden.

Zur Berechnung der Durchlaufzeit einer Instanz wird im ersten Schritt das State-Log mithilfe von `getStateLogToState` gebildet (vgl. 2.2.5 Logging). Diese Funktion liefert das State-Log eines MBAs bis zu einem bestimmten Zustand. Dazu bildet sie zuerst das gesamte State-Log. Da dieses zeitlich geordnet ist, d.h. Zustände die früher aktiv waren als andere stehen vor Zuständen, die erst später aktiv waren, kann es mithilfe der XPath-Achsen `self` und `preceding-sibling` gefiltert werden [Clark und DeRose, 2015, #axes]. Dabei werden all jene Zustände entfernt, die nach dem ersten Auftreten des Zustands mit der Bezeichnung `$toState` betreten wurden. Wurde kein Wert für `$toState` übergeben, wird das gesamte State-Log zurückgeliefert. Für den Fall, dass `$toState` noch nicht erreicht wurde, retourniert die Funktion eine leere Menge. Um schließlich die Durchlaufzeit berechnen zu können, muss zwischen drei Fällen unterschieden werden. Der erste Fall deckt die Verwendung des Parameters `$toState` ab. Um die Dauer bis zur Erreichung dieses Zustands zu erhalten, wird das `from`-Attribut des ersten Eintrags im State-Log vom `from`-Attribut des letzten Eintrags subtrahiert. Beim zweiten und dritten Fall wurde für den Parameter `$toState` eine leere Menge übergeben, daher muss die gesamte Durchlaufzeit des MBAs berechnet werden. Hier ist es möglich, dass der Prozess entweder bereits abgeschlossen oder noch aktiv ist. Einen abgeschlossenen Prozess erkennt man dadurch, dass der letzte Eintrag im State-Log über ein `until`-Attribut verfügt. Die Durchlaufzeit ist in diesem Fall die Dauer zwischen dem `from` des ersten und dem `until`-Attribut des letzten State-Log Eintrags. Sollte das MBA zum Zeitpunkt der Analyse noch

Quellcode 18: Funktion getCycleTimeOfInstance

```
1  declare function analysis:getCycleTimeOfInstance(
2      $mba as element(),
3      $toState as xs:string?
4  ) as xs:duration? {
5    let $stateLog :=
6      analysis:getStateLogToState($mba, $toState)/state
7
8    return
9      let $cycleTimeOfInstance :=
10       (if ($toState) then
11         xs:dateTime($stateLog[last()]/@from)
12       else if ($stateLog[last()]/@until) then
13         xs:dateTime($stateLog[last()]/@until)
14       else
15         fn:current-dateTime()
16       ) - xs:dateTime(($stateLog/@from)[1])
17 };
```

aktiv sein, würde das `until`-Attribut des letzten Eintrags fehlen und es muss auf die aktuelle Systemzeit zurückgegriffen werden.

Quellcode 19 zeigt die Funktion `getAverageCycleTime`, welche zur Berechnung der durchschnittlichen Durchlaufzeit eines einzelnen Zustands dient. Sie wurde von [Schuetz, 2015, S. 206] übernommen und erweitert. Die Durchlaufzeit wird berechnet, indem über alle Konkretisierungen der angegebenen Ebene iteriert und die Zeiten des gesuchten Zustands (`$stateId`) in einer Sequenz gespeichert werden. Über diese wird abschließend der Durchschnitt mithilfe der Built-In Funktion `avg` gebildet. Hervorzuheben ist die Summenbildung zwischen den Zeilen 15 und 20. Der FLWOR-Ausdruck als erster Parameter der Summenfunktion ist notwendig, da Zustände durch Schleifen im Prozess mehrmals aktiv werden können, wodurch es möglich ist dass es mehrere Einträge für einen Zustand im State-Log gibt. Der zweite Parameter definiert einen alternativen Nullwert, der verwendet wird falls das erste Argument eine leere Menge ist [W3C, 2014a, #func-sum]. Damit der Durchschnitt korrekt berechnet werden kann, müssen leere Mengen als solche in die Sequenz `$cycleTimes` übernommen werden, da leere Mengen von der `avg`-Funktion ignoriert werden. Eine Verwendung einer Dauer von null (0) anstelle der leeren Menge würde das Ergebnis verfälschen.

Um abzufragen, wie lange sich ein MBA in ausgewählten Zuständen befindet, wird die Funktion `getTotalCycleTimeInStates` verwendet, zu sehen in Quellcode 20. Die Berechnung der Summe erfolgt analog zu `getAverageCycleTime`, mit dem Unterschied, dass anstelle einer Zustandsbezeichnung (`$stateId`) eine Sequenz von Zustandsbezeichnungen übergeben wird. Dadurch muss für den

Quellcode 19: Funktion getAverageCycleTime

```
1  declare function analysis:getAverageCycleTime(
2      $mba as element(),
3      $level as xs:string,
4      $inState as xs:string?,
5      $stateId as xs:string
6  ) as xs:duration? {
7    let $descendants :=
8      analysis:getDescendantsAtLevelOrMBA($mba, $level)
9      [if ($inState) then mba:isInState(., $inState) else true()]
10
11   let $cycleTimes :=
12     for $descendant in $descendants
13       let $stateLog := analysis:getStateLog($descendant)
14       return
15         fn:sum(
16           for $entry in $stateLog/state[@ref = $stateId]
17           return
18             analysis:getCycleTimeOfStateLogEntry($entry)
19           ,
20           ()
21         )
22
23   return fn:avg($cycleTimes)
24 };
```

Quellcode 20: Funktion getTotalCycleTimeInStates

```
1  declare function analysis:getTotalCycleTimeInStates(
2      $mba as element(),
3      $states as xs:string*
4  ) as xs:duration? {
5    let $stateLog := analysis:getStateLog($mba)
6
7    return fn:sum(
8      (
9        for $entry in $stateLog/state[
10               functx:is-value-in-sequence(@ref, $states)
11           ]
12       return
13         analysis:getCycleTimeOfStateLogEntry($entry)
14     ),
15     ()
16   )
17 };
```

Vergleich der Bezeichnungen die Built-In Funktion `is-value-in-sequence` herangezogen werden. Diese hat zwei Parameter, der erste beinhaltet den gesuchten Wert und der zweite die Sequenz in der die Suche durchgeführt werden soll. Sie liefert „true“, falls sich der gesuchte Wert in der Sequenz befindet und ansonsten „false“.

In Quellcode 21 wird die Funktion `getTotalCycleTime` gezeigt, mit der die What-If-Analyse über den gesamten Prozess eines MBAs durchgeführt werden kann. Die Durchlaufzeit wird berechnet, indem die Durchlaufzeiten aller Zustände des Prozesses nach ihren Wahrscheinlichkeiten gewichtet und aufsummiert werden. Damit keine Zustände mehrfach gezählt werden, erfolgt die Iteration nur über die abstraktesten Zustände, d.h. alle `state`-, `parallel`- und `final`-Elemente des SCXML-Modells, die keinem anderen Zustand untergeordnet sind. In Abbildung 19 sind das beispielsweise die Zustände „ChooseProducts“, „DevelopProduct“ und „Finished“. Mithilfe des Parameters `$excludeArchiveStates` können mit dem Attribut `mba:isArchiveState` gekennzeichnete Endzustände von der Auswertung ausgeschlossen werden. Die Durchlaufzeiten der Zustände werden von der Funktion `getCycleTimeForCompositeState` berechnet, beschrieben in Abschnitt 3.2.2.

Die zweite XQuery-Funktion des Moduls mit der What-If-Analysen durchgeführt werden können ist `getTotalCycleTimeToState`, abgebildet in Quellcode 22. Der große Unterschied im Vergleich zur vorhergehenden Funktion `getTotalCycleTime` ist, dass anstatt der gesamten Durchlaufzeit jene Dauer berechnet wird, bis ein festgelegter Zustand erreicht wurde, wobei die Dau-

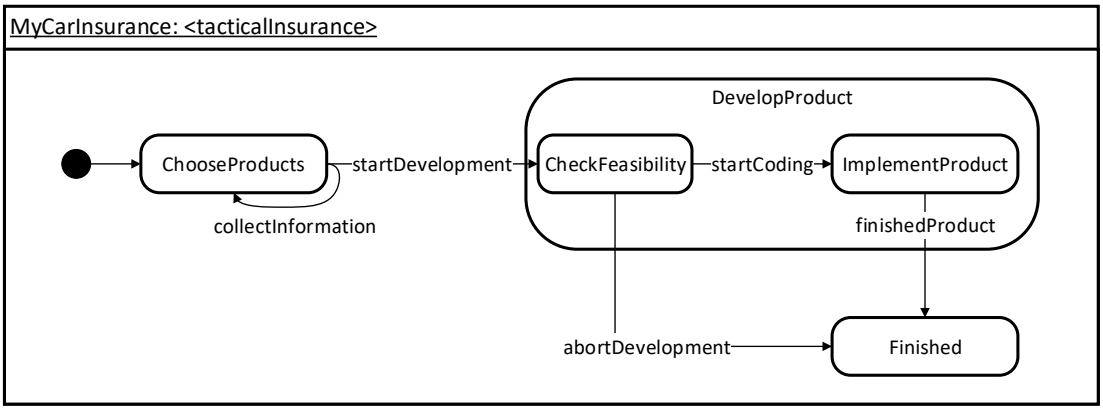

Abbildung 19: Beispielprozess eines MBAs

Quellcode 21: Funktion getTotalCycleTime

```
declare function analysis:getTotalCycleTime(
    $mba as element(),
    $level as xs:string,
    $inState as xs:string?,
    $excludeArchiveStates as xs:boolean?,
    $changedStates as element()?,
    $changedTransitions as element()*,
    $changedTransitionsFactors as xs:decimal*
) as xs:duration {
  let $scxml := analysis:getSCXMLAtLevel($mba, $level)

  let $states :=
    analysis:getMostAbstractStates($scxml, $excludeArchiveStates)

  return
    sum(
      for $state in $states
      return
        analysis:getCycleTimeForCompositeState(
          $mba, $level, $inState, $state,
          $changedStates,
          $changedTransitions, $changedTransitionsFactors)
    )
};
```

er dieses Zustands nicht in die Kalkulation miteinbezogen wird. Die zur Berechnung verwendeten Algorithmen unterscheiden sich dabei grundlegend voneinander. Während bei `getTotalCycleTime` die Durchlaufzeiten sämtlicher Zustände zusammengerechnet werden kann, muss bei der Auswertung eines Prozesses bis zu einem bestimmten Zustand beachtet werden, dass dieser über verschiedene Pfade erreichbar sein kann. Die Funktion durchläuft den Prozess, ausgehend vom Zielzustand `$toState`, von hinten nach vorne und folgt allen möglichen Pfaden bis zum Startzustand des Prozesses. Dieser ist ein `initial`-Element, das direkt unter dem `scxml`-Element definiert ist. Ein Prozessmodell eines MBAs verfügt immer über genau einen Startzustand.

Zu Beginn der Funktion wird der Ausgangszustand (`$toState`) aus dem SCXML-Modell des MBAs selektiert und überprüft, ob er sich in einer SZK befindet (vgl. 2.3.1 Quantitative Analyse). Die Funktion `getStateList` (siehe Zeile 25 in Quellcode 22) liefert von dem übergebenen Zustand `$source` ausgehend alle Pfade bis zum Startzustand des MBAs in Form einer Liste von Zuständen mit ihren Durchlaufzeiten, welche bereits nach den entsprechenden Transitionswahrscheinlichkeiten bzw. den übergebenen Parametern (`$changedStates` `$changedTransitions`, `$changedTransitionsFactors`) gewichtet sind. Das Prozessmodell wird dazu von hinten nach vorne durchlaufen. Die leere Menge als letzter Parameter des Aufrufs ist der Startwert einer Liste, in der alle bisher besuchten Zustände gespeichert werden und die daher zu Beginn leer sein muss. Sie verhindert redundante Aufrufe und wird benötigt, um Schleifen berechnen zu können. Quellcode 23 zeigt ein Beispiel für den Rückgabewert dieser Funktion.

Die Funktion `getStateList` ist in Quellcode 24 dargestellt. Falls sich der übergebene Zustand (`$state`) bereits in der Liste `$stateList` befindet, muss er nicht weiter bearbeitet werden und `$stateList` wird unverändert zurückgegeben. Hier ist anzumerken, dass Initialzustände (`initial`) über kein `id`-Attribut verfügen. In diesem Fall liefert die Überprüfung in Zeile 13 ebenfalls „true“ und der erste `if`-Zweig wird betreten. Anschließend wird die Durchlaufzeit berechnet, dies erfolgt allerdings nur, wenn es sich bei dem Zustand weder um ein `initial`-Element noch den Zielzustand (`$toState`) handelt. Falls `$cycleTime` mit einem Wert ungleich der leeren Menge initialisiert wurde, wird der Listeneintrag für den Rückgabewert erstellt, welcher anschließend an den Parameter `$stateList` angehängt wird.

Ab Zeile 33 werden die Transitionen selektiert, die dazu führen, dass der gesuchte Zustand `$state` aktiv wird (vgl. 3.2.3 Bestimmung eingehender Transitionen). Die Ausgangszustände dieser Transitionen sind Vorgänger des Zustands `$state`, d.h. er kann von ihnen erreicht werden. Daher müssen sie in die Berechnung der Zeit, bis dieser Zustand erreicht wird, miteinbezogen werden. Befindet sich der Zielzustand (`$toState`) in einer Schleife des Prozessmodells (d.h. der Parameter

Quellcode 22: Funktion getTotalCycleTimeToState

```
1  declare function analysis:getTotalCycleTimeToState(
2      $mba as element(),
3      $level as xs:string,
4      $inState as xs:string?,
5      $toState as xs:string,
6      $changedStates as element()?,
7      $changedTransitions as element()*,
8      $changedTransitionsFactors as xs:decimal*
9  ) as xs:duration {
10   let $scxml := analysis:getSCXMLAtLevel($mba, $level)
11   let $state := $scxml//*[@id = $toState]
12   let $sccMap := analysis:tarjanAlgorithm($scxml)
13   let $scc :=
14     fn:for-each(
15       map:keys($sccMap),
16         function($k){
17           if (functx:is-node-in-sequence(
18             $state,
19             map:get($sccMap, $k))
20           ) then
21             map:get($sccMap, $k)
22           else ()
23         }
24     )
25   let $stateList :=
26     analysis:getStateList($mba, $level, $inState, $state,
27       $changedStates,
28       $changedTransitions, $changedTransitionsFactors,
29       $toState, $scc, ())
30   return
31     fn:sum(
32       for $state in $stateList
33       let $parentStates :=
34         analysis:getParentStates($scxml, $state/@id)
35       return
36         if (not($parentStates = $stateList/@id)) then
37           xs:dayTimeDuration($state/@averageCycleTime)
38         else ()
39     )
40 };
```

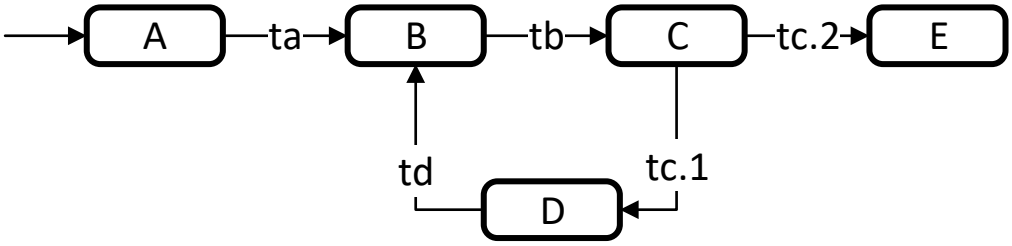

Abbildung 20: Prozessmodell mit Schleife

`$scc` ist ungleich der leeren Menge), dürfen nicht alle Vorgänger verwendet werden. Dieses Problem wird in Abbildung 20 verdeutlicht. Sie zeigt ein Prozessmodell in dem sich die Zustände „B“, „C“ und „D“ in einer stark zusammenhängenden Komponente (SZK) befinden und somit eine Schleife bilden. Soll die Durchlaufzeit berechnet werden bis der Zustand „E“ aktiv wird, müssen die Durchlaufzeiten seiner Vorgänger addiert werden, d.h. die der Zustände „C“, „B“, „D“ und „A“. Wird als Zielzustand allerdings „C“ gewählt, d.h. ein Zustand aus der SZK, müssen die Vorgänger eingeschränkt werden. „C“ kann nur über „B“ direkt erreicht werden. Dieser hat die zwei Vorgänger „A“ und „D“, allerdings ist letzterer wiederum ein Vorgänger des gesuchten Zielzustands „C“. Dieser kann erst aktiv werden, nachdem „C“ bereits verlassen wurde und muss deshalb von der Berechnung ausgeschlossen werden. Die Funktion `getStateList` löst dieses Problem, indem, falls sich der Zielzustand in einer SZK befindet, Transitionen von einem Zustand in der SZK zu ihrem Wurzelknoten nicht verfolgt werden. Ist der Zustand kein Teil einer Schleife, wird allen eingehenden Transitionen nachgegangen. Im `return`-Teil der Funktion wird mithilfe der Built-In Funktion `fold-left` [W3C, 2014a, #func-fold-left] die Liste von Zuständen iterativ zusammengesetzt, indem die Funktion `getStateList` für die Ausgangszustände der zuvor selektierten Transitionen rekursiv aufgerufen wird. Der Parameter `$stateList` wird bei jeder Iteration aktualisiert, so dass die Durchlaufzeit eines Zustand nur einmal berechnet wird und keine Duplikate zurückgeliefert werden.

Abschließend wird in `getTotalCycleTimeToState` die Durchlaufzeit durch eine Iteration über die Liste von Zuständen berechnet. Dabei muss allerdings geprüft werden, ob sich Zustände in der Liste befinden, die in einer hierarchischen Beziehung zueinander stehen. Beinhaltet sie beispielsweise einen Zustand und seine Subzustände, dürfen die Subzustände nicht in das Ergebnis mit eingerechnet werden, da ihre Durchlaufzeit bereits in der des übergeordneten Zustands enthalten ist. Die Funktion `getParentStates` selektiert vom übergebenen Zustand mithilfe der XPath-Achse `ancestor` [Clark und DeRose, 2015, #axes] alle übergeordneten Zustände (`state`, `parallel`, `final`) und lieferte eine Sequenz ihrer Zustandsbezeichner (`id`) zurück. In die gesamte Durchlaufzeit werden nur

Zustände miteinbezogen, die keinem anderen Knoten in der Liste untergeordnet sind.

Quellcode 23: Rückgabewert der Funktion getStateList

```
1 <state id="DevelopProducts" averageCycleTime="PT18M20S"/>
2 <state id="ChooseProducts" averageCycleTime="PT16M40S"/>
```

Quellcode 24: Funktion getStateList

```
1  declare function analysis:getStateList(
2      $mba as element(),
3      $level as xs:string,
4      $inState as xs:string?,
5      $state as element(),
6      $changedStates as element()?,
7      $changedTransitions as element()*,
8      $changedTransitionsFactors as xs:decimal*,
9      $toState as xs:string,
10     $scc as element()*,
11     $stateList as element()*
12 ) as element()* {
13   if (not($stateList[@id = $state/@id])) then
14     let $cycleTime :=
15       if (not(analysis:stateIsInitial($state)) and
16         fn:compare($state/@id, $toState) != 0) then
17         analysis:getCycleTimeForCompositeState(
18           $mba, $level, $inState, $state, $changedStates,
19           $changedTransitions, $changedTransitionsFactors)
20       else
21         ()
22     let $entry :=
23       if (not(empty($cycleTime))) then
24         element {'state'} {
25           $state/@id,
26           attribute averageCycleTime {$cycleTime}
27         }
28       else
29         ()
30     let $stateList := ($stateList, $entry)
31     let $scxml := analysis:getSCXMLAtLevel($mba, $level)
32     let $transitions :=
33       let $t := analysis:getTransitionsToState($scxml, $state)
34       return
35         if ($scc and
36           ($state is analysis:getRootNodeOfSCC($scc))) then
37           $t[not(
38             functx:is-node-in-sequence(sc:getSourceState(.), $scc)
39             )]
40         else
```

```
41            $t
42      return
43        fn:fold-left(
44          $transitions,
45          $stateList,
46          function($result, $t) {
47            let $source := sc:getSourceState($t)
48            return
49              analysis:getStateList(
50                $mba, $level, $inState, $source, $changedStates,
51                $changedTransitions, $changedTransitionsFactors,
52                $toState, $scc, $result)
53          })
54    else
55      $stateList
56  };
```

3.2.2 Durchschnittliche Durchlaufzeit zusammengesetzter Zustände

Die XQuery-Funktion zur Berechnung der Durchlaufzeit von zusammengesetzten Zuständen ist in Quellcode 25 zu sehen. Sie ist rekursiv aufgebaut und ermittelt aufgrund von Daten aus Event- bzw. State-Logs die zu erwartende Durchlaufzeit eines Zustands. Es wird davon ausgegangen, dass sich die Durchlaufzeit eines zusammengesetzten Zustands grundsätzlich aus der Summe der Durchlaufzeiten seiner Subzustände ergibt. Eine Ausnahme davon stellen parallele Zustände dar. Bei diesen ist die Durchlaufzeit gleich der des am längsten dauernden parallelen Pfads.

Die Schnittstelle der Funktion beschreibt sieben Parameter. Die ersten beiden, `$mba` und `$level`, identifizieren das MBA bzw. die zu untersuchende Ebene. Über `$inState` können Benutzerinnen und Benutzer einschränken, dass nur MBAs, die sich aktuell in einem bestimmten Zustand befinden, in die Analyse miteinbezogen werden. Der Parameter `$state` beinhaltet den Zustand, für den die Durchlaufzeit berechnet werden soll. Die letzten drei Parameter ermöglichen entsprechend der What-If-Analyse das Verändern von Durchlaufzeiten bzw. Transitionswahrscheinlichkeiten. Sie werden in Abschnitt 3.1.2 genauer erläutert.

Prinzipiell unterscheidet die Funktion `getCycleTimeForCompositeState` zwei Fälle, entweder ein Subzustand des zu berechnende Zustand befindet sich in der Liste der zu verändernden Zustände oder nicht. Daher muss für den ersten Fall geprüft werden, ob sich die Bezeichnung (`id`) eines Subzustands des zu berechnenden Zustands (`$state`) in `$changedStates` befindet. Trifft dieser Fall zu, muss die Durchlaufzeit für alle Subzustände von `$state`, die sich genau eine Ebene unter ihm befinden, über rekursive Aufrufe von `getCycleTimeForCompositeState` separat berechnet werden. Entsprechend den XPath-Achsen

werden diese als Kind-Knoten bezeichnet [Clark und DeRose, 2015, #axes]. Die sich ergebenden Durchlaufzeiten werden in einer Liste zwischengespeichert. Um die korrekte Zeit zu bestimmen, muss noch einmal eine Unterscheidung aufgrund der Art des Zustands vorgenommen werden. Handelt es sich bei ihm um ein `parallel`-Element ist die Durchlaufzeit des Zustands die maximale Dauer in der Liste, ansonsten die Summe.

Wenn kein Subzustand von `$state` verändert werden soll, wird der `else`-Zweig betreten, zu sehen ab Zeile 25 in Quellcode 25. Entsprechend der *Flow Analysis* ergibt sich die durchschnittliche Durchlaufzeit eines Zustands aus seiner Durchlaufzeit (`getAverageCycleTime`) multipliziert mit der Wahrscheinlichkeit, mit der er in einem Prozess aktiv wird (`getTransitionProbabilityForTargetState`). Da die Funktion `getCycleTimeForCompositeState` im Zuge der What-If-Analyse eine manuelle Gewichtung der Zeiten eines Zustands ermöglicht, muss das Zwischenergebnis noch mit dem entsprechenden Faktor aus `$changedStates` multipliziert werden.

Quellcode 25: Funktion getCycleTimeForCompositeState

```
1  declare function analysis:getCycleTimeForCompositeState(
2      $mba as element(),
3      $level as xs:string,
4      $inState as xs:string?,
5      $state as element(),
6      $changedStates as element()?,
7      $changedTransitions as element()*,
8      $changedTransitionsFactors as xs:decimal*
9  ) as xs:duration? {
10   let $scxml := analysis:getSCXMLAtLevel($mba, $level)
11   return
12     if ($state/(descendant::sc:state | descendant::sc:parallel |
13         descendant::sc:final)/@id = $changedStates/state/@id) then
14       let $cycleTimes :=
15         for $substate in $state/(sc:state | sc:parallel | sc:final)
16         return
17           analysis:getCycleTimeForCompositeState($mba, $level,
18             $inState, $substate, $changedStates,
19             $changedTransitions, $changedTransitionsFactors)
20       return
21         if (analysis:isParallel($state)) then
22           max($cycleTimes)
23         else
24           fn:sum($cycleTimes)
25     else
26       let $sccMap := analysis:tarjanAlgorithm($scxml)
27       return
28         analysis:getAverageCycleTime($mba, $level, $inState,
29           $state/@id) *
30         analysis:getTransitionProbabilityForTargetState(
31           $scxml, $state, $changedTransitions,
32           $changedTransitionsFactors, $sccMap) *
33         analysis:getChangedStateFactor($state, $changedStates)
34 };
```

3.2.3 Bestimmung eingehender Transitionen

Bei einigen Funktionen im Analysemodul müssen alle eingehenden Transitionen eines Zustands bestimmt werden. Das Analysemodul stellt diese Funktionalität mithilfe der beiden überladenen Funktionen `getTransitionsToState` bereit, die sich durch die Anzahl ihrer Parameter unterscheiden.

Quellcode 26 zeigt die erste Variante mit zwei Parametern. Sie ruft die zweite, rekursive Funktion `getTransitionsToState` auf und setzt die dafür erforderlichen Parameter `$includeSubstates` und `$checkParallel` auf „true“, da diese Konfiguration für den ersten Aufruf benötigt wird.

Quellcode 26: Funktion getTransitionsToState mit zwei Parametern

```
1 declare function analysis:getTransitionsToState(
2     $scxml as element(),
3     $state as element()
4 ) as element()* {
5   analysis:getTransitionsToState($scxml, $state, true(), true())
6 };
```

Die zweite Variante ist in Quellcode 27 dargestellt. Sie unterscheidet zwischen vier Fällen. Die ersten beiden betreffen Initialzustände, der dritte behandelt parallele Zustände und der vierte bearbeitet alle übrigen Varianten. Im letzten Fall wird aufgrund des Parameters `$includeSubstates` unterschieden. Dieser führt dazu, dass anstatt aller Transitionen, die dazu führen, dass der Zustand aktiv wird, nur Transitionen inkludiert werden die diesen als Ziel haben. Ist `$includeSubstates` auf „true“ gesetzt, werden alle Transitionen selektiert deren Attribut `target` entweder der Bezeichnung des Zustands selbst oder der einem seiner Subzustände entspricht. Anschließend werden über einen XPath-Ausdruck noch jene Transitionen entfernt, bei denen der Ausgangszustand ebenfalls ein Subzustand von `$state` ist, da dieser bereits vor dem Auftreten der Transition aktiv gewesen sein muss. Sollen keine Subzustände inkludiert werden, reicht die Selektion aller Transitionen des `scxml`-Elements mit der Bezeichnung des Zustands im `target`-Attribut. Ab Zeile 42 folgt die Abhandlung von `initial`-Attributen, über die Zustände verfügen können. Falls das dem Zustand übergeordnete Element auf diese Art und Weise `$state` als Initialzustand deklariert, müssen Transitionen auf diesen übergeordneten Zustand ebenfalls miteinbezogen werden.

Bei Initialzuständen, die als `initial`-Elemente deklariert wurden (siehe Zeilen 7 bis 11 in Quellcode 27), wird zuerst geprüft ob es sich um den Initialzustand des Prozessmodells handelt. Das trifft zu, wenn sich das `initial`- direkt unter dem `scxml`-Element befindet. In diesem Fall ist der Zustand über keine Transition erreichbar und es wird eine leere Menge zurückgegeben. Auch Zustände können Initialzustände besitzen, diese werden durch Transitionen auf das übergeordnete Element betreten. Daher muss, falls es sich bei `$state` um den Initialzustand ei-

Quellcode 27: Funktion getTransitionsToState mit vier Parametern

```
1  declare function analysis:getTransitionsToState(
2      $scxml as element(),
3      $state as element(),
4      $includeSubstates as xs:boolean,
5      $checkParallel as xs:boolean
6  ) as element()* {
7    if (analysis:stateIsInitialOfSCXML($state)) then
8      ()
9    else if (analysis:stateIsInitial($state)) then
10     analysis:getTransitionsToState($scxml, $state/..,
11       false(), true())
12   else if (analysis:parentIsParallel($state) and
13     $checkParallel) then
14     (
15       analysis:getTransitionsToState($scxml, $state/..,
16         $includeSubstates, true())
17       ,
18       if (not($includeSubstates)) then
19         let $siblings := analysis:getSiblingStates($state)
20         return
21           (
22             for $s in $siblings
23             return analysis:getTransitionsToState($scxml, $s,
24               true(), false())
25             ,
26             analysis:getTransitionsToState($scxml, $state,
27               $includeSubstates, false())
28           )
29       else ()
30     )
31   else
32     (
33       if ($includeSubstates) then
34         let $substates :=
35           analysis:getStateAndSubstates($scxml, $state/@id)
36         return
37           $scxml//sc:transition[@target = $substates]
38             [not(functx:is-value-in-sequence(../@id, $substates))]
39       else
40         $scxml//sc:transition[@target = $state/@id]
41       ,
42       if (fn:compare(fn:string($state/../@initial),
43         fn:string($state/@id)) = 0) then
44         analysis:getTransitionsToState($scxml, $state/..,
45           false(), true())
46       else ()
47     )
48 };
```

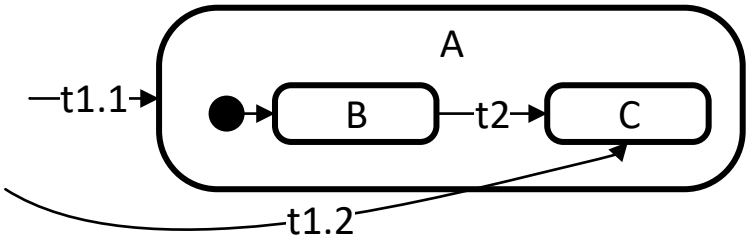

Abbildung 21: Initialzustand eines `state`-Elements

nes Zustands handelt, `getTransitionsToState` mit dem auf „false“ gesetzten Parameter `$includeSubstates` erneut aufgerufen werden. Abbildung 21 illustriert diesen Sachverhalt. Der Zustand „A“ besitzt zwei Subzustände „B“ und „C“ sowie einen Initialzustand. Der Initialzustand wird durch die Transition „t1.1“ aktiv, d.h. durch eine Transition mit dem Ziel „A“. Die Transition „t1.2“ führt dazu, dass „C“ und damit auch der übergeordnete Zustand „A“ aktiv werden, allerdings überspringt sie den Initialzustand sowie „B“.

Die Subelemente eines `parallel`-Elements werden gleichzeitig betreten, sobald dieses aktiv ist, d.h. eine Transition direkt zum `parallel`-Element oder zu einem ihrer Subzustände genommen wurde. Die Bestimmung von Transitionen zu parallelen Zuständen erfolgt ab Zeile 12 in Quellcode 27. Zuerst wird geprüft, ob das übergeordnete Element des gewünschten Zustands vom Typ `parallel` ist. Alle Transitionen, die entweder das dem Zustand (`$state`) übergeordnete Element oder einen ihrer Subzustände als Ziel haben, führen dazu, dass `$state` aktiv wird. Abbildung 22 zeigt einen Ausschnitt aus einem Prozessmodell mit parallelen Zuständen. Wird die Funktion `getTransitionsToState` für den Zustand „A“ aufgerufen, liefert sie alle drei Transitionen „t1“, „t2“ und „t3“ zurück, da „A“ direkt unter dem `parallel`-Element („P“) liegt und jede Transition entweder „P“ bzw. einen seiner Subzustände als Ziel hat. Das gilt allerdings nur für den Fall, dass der Parameter `$includeSubstates` auf „false“ gesetzt wurde. Bei rekursiven Aufrufen der Funktion aufgrund eines Initialzustands (siehe Zeile 10 bzw. 44) wird dieser Parameter auf „false“ gesetzt. Bei dem Beispiel aus Abbildung 22 passiert das unter anderem wenn die Funktion für den Initialzustand von „B“ aufgerufen wird. Aufgrund vom Typ des Zustands (`initial`) folgt ein erneuter Aufruf von `getTransitionsToState` für den übergeordneten Zustand („B“) mit dem Parameter `$includeSubstates` als „false“. Die Transitionen, die nun zurückgeliefert werden sollen sind jene, die entweder direkt zum übergeordneten `parallel`-Element („P“), einem Zustand auf der gleichen Ebene inklusiver Subzustände („A“, „A’“, „A”“) oder dem Zustand selbst („B“) ohne Subzustände gehen, d.h. „t1“ und „t2“.

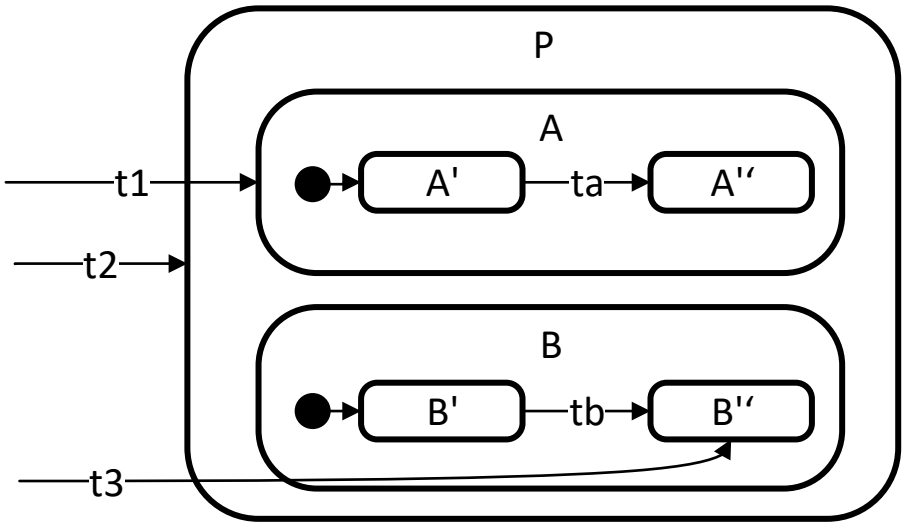

Abbildung 22: Transitionen zu einem `parallel`-Element

Der rekursive Aufruf von `getTransitionsToState` in Zeile 15 in Quellcode 27 berechnet den ersten Teil der Transitionen. Die zweite Gruppe wird in den Zeilen 22 bis 24 bestimmt. Mithilfe der Hilfsfunktion `getSiblingStates` werden alle Zustände auf der gleichen Ebene wie der übergebene Zustand `$state` selektiert und für jeden erfolgt ein rekursiver Aufruf von `getTransitionsToState`. Um zu verhindern, dass in weiterer Folge für diese Zustände erneut `getTransitionsToState` aufgerufen wird, wurde der vierte Parameter, `$checkParallel`, eingeführt. Dieser wird bei den Funktionsaufrufen auf „false“ gesetzt und verbietet eine erneute Überprüfung ob der Zustand `$state`- einem `parallel`-Element untergeordnet ist. Abschließend wird in Zeile 26 die dritte Gruppe bestimmt, indem alle Transitionen, die direkt auf den Zustand `$state` gehen, zurückgeliefert werden.

3.2.4 Implementierung der Schleifenerkennung

Das Analysemodul verwendet, wie in Abschnitt 2.3.4 beschrieben, den Algorithmus von Tarjan um Schleifen in Prozessen zu erkennen. Dieses Kapitel beschreibt die Umsetzung des Algorithmus in XQuery.

Die Funktion `tarjanAlgorithm`, zu sehen in Quellcode 28, berechnet alle stark zusammenhängende Komponenten (SZKs) eines Prozessmodells und liefert all jene zurück, die mehr als einen Zustand beinhalten. Für die Rückgabe verwendet sie eine Map-Datenstruktur (`map(*)`), bei der jeder Eintrag eine SZK repräsentiert. Die Einträge bestehen dabei aus einer Sequenz von Zuständen (`state`, `parallel`, `final`).

Quellcode 28: Funktion tarjanAlgorithm

```
1  declare function analysis:tarjanAlgorithm(
2      $scxml as element()
3  ) as map(*) {
4    let $scc :=
5      map:get(
6        fn:fold-left(
7          $scxml//(sc:state | sc:parallel | sc:final),
8          analysis:createMap(0, (), (), (), ()),
9          function($result, $state) {
10           let $indexes := map:merge(map:get($result, 'indexes'))
11           return
12             if (analysis:notVisited($indexes, $state/@id)) then
13               analysis:strongconnect($scxml, $state, $result)
14             else
15               $result
16         }
17       )
18       ,
19       'scc'
20     )
21   return
22     map:merge((
23       for $s in 0 to (map:size($scc) - 1)
24       let $item := map:get($scc, $s)
25       return
26         if (fn:count($item) > 1) then
27           map:entry($s, $item)
28         else
29           ()
30     ))
31 };
```

Damit alle Zustände besucht werden, iteriert eine `fold`-Funktion [W3C, 2014a, #func-fold-left] über eine Selektion aller `state`-, `parallel`- und `final`-Elemente des SCXML-Modells. Die benötigten Umgebungsvariablen wie z.B. Index und *Lowlink* jedes Zustands werden in einer Map gespeichert, die von der Funktion `createMap` erstellt wird. Zu Beginn wird diese mit leeren Werten befüllt (Zeile 8). Der erste Parameter entspricht dem nächsten zu vergebenden Index, der zweite einer Map für die bereits vergebenen Indizes und der dritte beinhaltet die *Lowlinks*. Der vierte Parameter simuliert einen Stack, auf dem die Zustände in der Reihenfolge, in der sie besucht wurden, abgelegt werden. Die gefundenen SZKs werden in einer Map im letzten Parameter abgelegt.

Quellcode 29: Funktion strongconnect

```
1  declare function analysis:strongconnect(
2      $scxml as element(),
3      $state as element(),
4      $resultMap as map(*)
5  ) as map(*) {
6    let $index := map:get($resultMap, 'index')
7    let $indexes := map:merge((
8        map:get($resultMap, 'indexes'),
9        map:entry($state/@id, $index)
10     ))
11   let $lowlink := $index
12   let $lowlinks := map:merge((
13       map:get($resultMap, 'lowlinks'),
14       map:entry($state/@id, $lowlink)
15     ))
16   let $stack := (map:get($resultMap, 'stack'), $state)
17   let $scc := map:get($resultMap, 'scc')
18   let $index := $index + 1
19   let $successors := analysis:getSuccessors($scxml, $state/@id)
20
21   let $recursiveResult :=
22     fn:fold-left(
23       $successors,
24       analysis:createMap($index, $indexes, $lowlinks, $stack, $scc),
25       function($result, $successor) {
26         if (analysis:notVisited($indexes, $successor/@id)) then
27           let $sucResult :=
28             analysis:strongconnect($scxml, $successor, $result)
29           let $lowlinks := map:get($sucResult, 'lowlinks')
30           let $lowlink := min((
31               map:get($lowlinks, $state/@id),
32               map:get($lowlinks, $successor/@id)
33             ))
34           return
35             analysis:createMap(
36               map:get($sucResult, 'index'),
37               map:get($sucResult, 'indexes'),
38               map:merge(($lowlinks, map:entry($state/@id,$lowlink))),
39               map:get($sucResult, 'stack'),
40               map:get($sucResult, 'scc')
41             )
42         else if (analysis:onStack($stack, $successor)) then
43           let $indexes := map:get($result, 'indexes')
44           let $lowlinks := map:get($result, 'lowlinks')
45           let $lowlink := min((
46               map:get($lowlinks, $state/@id),
47               map:get($indexes, $successor/@id)
48             ))
```

```
49          return
50            analysis:createMap(
51              map:get($result, 'index'),
52              map:get($result, 'indexes'),
53              map:merge(($lowlinks, map:entry($state/@id,$lowlink))),
54              map:get($result, 'stack'),
55              map:get($result, 'scc')
56            )
57        else
58          $result
59      }
60    )
61
62  let $index := map:get($recursiveResult, 'index')
63  let $indexes := map:get($recursiveResult, 'indexes')
64  let $lowlinks := map:get($recursiveResult, 'lowlinks')
65  let $stack := map:get($recursiveResult, 'stack')
66  let $scc := map:merge(map:get($recursiveResult, 'scc'))
67
68  return
69    if (map:get($lowlinks, $state/@id) =
70      map:get($indexes, $state/@id)) then
71      let $map := analysis:popStack($stack, (), $state)
72      let $newStack := map:get($map, 'stack')
73      let $newScc := map:get($map, 'scc')
74      return
75        analysis:createMap(
76          $index,
77          $indexes,
78          $lowlinks,
79          $newStack,
80          map:merge(($scc, map:entry(map:size($scc), $newScc)))
81        )
82    else
83      analysis:createMap($index, $indexes, $lowlinks, $stack, $scc)
84 };
```

Wurde ein Zustand noch nicht besucht, liefert die Funktion `notVisited` den Werte „true“ zurück und er wird durch Aufruf der Funktion `strongconnect` abgearbeitet. Sie benötigt das SCXML-Modell (`$scxml`), den zu untersuchenden Zustand (`$state`) und die Map mit den aktuellen Umgebungsvariablen (`$result`).

Die Funktion `strongconnect` ist in Quellcode 29 abgebildet. Zuerst wird der nächste Index von den Umgebungsvariablen geholt, und für den aktuellen Zustand ein neuer Eintrag in der `indexes`-Map angelegt. Analog dazu wird der *Lowlink* des Zustands mit dem Wert des Index initialisiert und ebenfalls in die entsprechende Map eingetragen. In Zeile 16 wird der aktuelle Zustand auf dem Stack abgelegt und anschließend der Wert für den nächsten zu verwendenden Index erhöht. Ausge-

hend vom aktuellen Zustand wird ab Zeile 22 mithilfe eines `folds` die Tiefensuche ausgeführt, indem über seine Nachfolger (`$successors`) iteriert wird. Für den zweiten Parameter des `folds` wird die Map für die Umgebungsvariablen mit den aktuellen Werten neu erzeugt.

Wurde ein Nachfolger noch nicht besucht, wird die Funktion `strongconnect` rekursiv aufgerufen, welche die aktualisierten Umgebungsvariablen zurückliefert. Anschließend muss der *Lowlink* des aktuellen Zustands (`$state`) neu gesetzt werden: wurde über den Nachfolger ein früherer Zustand, d.h. mit einem niedrigeren Index, erreicht, kann dieser vom aktuellen Zustand ebenfalls erreicht werden. Sein *Lowlink* ergibt sich aus dem Minimum seines eigenen *Lowlinks* und dem des Nachfolgers.

Falls der Nachfolger bereits besucht wurde und auf dem Stack liegt, bedeutet das, dass ein früherer Zustand und somit eine Schleife gefunden wurde. Der Nachfolger und der aktuelle Zustand befinden sich damit in der gleichen SZK. Nach dem Setzen des *Lowlinks* auf das Minimum vom aktuellen *Lowlink* und dem Index des Nachfolgers können die aktualisierten Umgebungsvariablen retourniert werden.

Nachdem alle Nachfolger eines Zustands abgearbeitet wurden, wird die Iteration beendet. Vor dem Verlassen der Funktion `strongconnect` wird noch geprüft, ob der aktuelle Zustand die Wurzel einer SZK ist, d.h. ob sein Index den gleichen Wert wie sein *Lowlink* aufweist. In diesem Fall werden mithilfe der Funktion `popStack` solange Zustände vom Stack geholt, bis der geholte Zustand gleich dem aktuellen ist. Sie liefert eine Map mit dem neuen Stack und einer Sequenz mit den Zuständen der SZK zurück. Diese werden in die Umgebungsvariablen gespeichert und abschließend von der Funktion `strongconnect` zurückgegeben.

3.2.5 Transitionswahrscheinlichkeiten

Die Funktion `getTransitionProbability`, zu sehen in Quellcode 30, berechnet die relative Wahrscheinlichkeit einer Transition. Der einzige Parameter dieser Funktion ist ein `transition`-Element aus dem `scxml`-Modell, das die gewünschte Transition repräsentiert. Die Funktion liefert die Wahrscheinlichkeit als Dezimalzahl vom Typ `xs:decimal` zurück. Sie iteriert dazu über die Event-Logs aller Konkretisierungen auf der entsprechende Ebene und fügt der Variable `$prob` für jedes Ereignis ein `log`-Element mit den Attributen `leftState` und `tookTransition` hinzu. In dem ersten Attribut ist vermerkt, ob das Ereignis eine Transition auslöste, die dazu führte, dass der Ausgangszustand der übergebenen Transition (`$transition`) verlassen wurde. Das zweite Attribut zeigt an, ob das Ereignis die übergebene Transition bzw. eine Spezialisierung davon auslöste. In der Zeile 6 von Quellcode 30 wird eine Überprüfung durchgeführt, ob der Ausgangszustand der Transition ein `initial`-Element ist. Da diese Elemente immer genau eine ausgehende Transition besitzen, ist deren Wahrscheinlichkeit per Definition 100%.

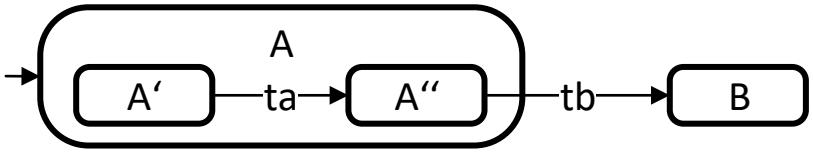

Abbildung 23: Verlassen von Zuständen durch Transitionen

Die Funktion weist daher der Variable `$prob` ein `log`-Element zu, bei dem die beiden Attribute `leftState` und `tookTransition` jeweils „true“ sind. Das wird im `return`-Teil der Funktion zu dem Dezimalwert `1.0` ausgewertet, da wie in den Zeilen 27 bis 30 ersichtlich die Anzahl aller Einträge, bei denen die Transition genommen wurde (`tookTransition="true"`) durch die Häufigkeit, mit die der Ausgangszustand verlassen wurde (`leftState="true"`) dividiert wird.

Um Spezialisierungen von Transitionen entsprechend behandeln zu können muss festgelegt werden, wann eine Transition dazu führt dass ein Zustand verlassen wird und wann eine Transition gleich bzw. eine Spezialisierung einer anderen ist. Die Funktion `stateIsLeft` liefert je nachdem ob ein Zustand durch eine Transition verlassen wird „true“ bzw. „false“ zurück. Dazu benötigt sie zum einen die Transition (`$transition`) und zum anderen die Bezeichnung des zu prüfenden Zustands (`$state`). Ein Zustand ist aktiv, solange er selbst bzw. mindestens ein Subzustand aktiv ist. Daher wird er durch eine Transition genau dann verlassen, wenn diese von ihm bzw. einem seiner Subzustände ausgeht und der Zielzustand weder der Zustand selbst noch einer seiner Subzustände ist. Abbildung 23 zeigt einen Ausschnitt aus einem Prozessmodell, der diesen Sachverhalt verdeutlicht. Der Zustand „A“, welcher die zwei Subzustände „A’“ und „A”“ besitzt, wird durch die Transition „tb“ verlassen, da die Transition von einem seiner Subzustände („A”“) ausgeht und der Zielzustand („B“) ist weder der Zustand selbst noch ein Subzustand. Anders verhält es sich mit der Transition „ta“. Diese geht zwar mit „A’“ von einem Subzutand von „A“ aus, allerdings ist das Ziel („A”“) ebenfalls einer, wodurch „A“ nicht verlassen wird und demnach aktiv bleibt. Ein Sonderfall tritt auf, falls eine Transition von „A“ nach „B“ geht und festgestellt werden soll, ob einer der Subzustände von „A“ durch diese Transition verlassen wird. Auf Basis der Informationen, die vom Modell bereitgestellt werden, kann das nicht eindeutig festgestellt werden, da zum Zeitpunkt des Auftretens der Transition entweder „A’“ oder „A”“ aktiv sein kann. Dieser Fall verletzt allerdings die in Abschnitt 2.2.3 beschrieben Regeln. Diese besagen, dass, falls ein Zustand durch Subzustände verfeinert wurde, nie der übergeordnete Zustand alleine aktiv sein darf.

Ob eine Transition gleich bzw. eine Spezialisierung einer anderen ist wird von der Funktion `compareTransitions` ermittelt. Gleich wie zuvor bei `stateIsLeft` ist der Rückgabewert vom Typ `xs:boolean`. Die Funktion benötigt zwei

Quellcode 30: Funktion getTransitionProbability

```
1  declare function analysis:getTransitionProbability(
2      $transition as element()
3  ) as xs:decimal {
4    let $sourceState := sc:getSourceState($transition)
5    let $prob :=
6      if (analysis:stateIsInitial($sourceState)) then
7        <log leftState='true' tookTransition='true'/>
8      else
9        let $mba := $transition/ancestor::mba:mba[last()]
10       let $level := $transition/ancestor::mba:elements[1]/../@name
11       let $descendants :=
12         analysis:getDescendantsAtLevelOrMBA($mba, $level)
13       for $descendant in $descendants
14       let $log := analysis:getEventLog(mba:getSCXML($descendant))
15       return
16         for $event in $log/xes:trace/xes:event
17         let $transitionEvent := analysis:getTransitionForLogEntry(
18           mba:getSCXML($descendant), $event)
19         return
20           <log leftState='{analysis:stateIsLeft(
21                 $transitionEvent, $sourceState/@id)
22               }'
23             tookTransition='{analysis:compareTransitions(
24                 $transition, $transitionEvent)
25               }'/>
26   return
27     if (fn:count($prob[@leftState = 'true']) > 0) then
28       fn:count(
29         $prob[@tookTransition = 'true' and @leftState = 'true']
30       ) div fn:count($prob[@leftState = 'true'])
31     else
32       0
33 };
```

Parameter, der erste ist die allgemeinere Transition und der zweite die speziellere. Bei den Aufrufen in der Funktion `getTransitionProbability` wird für die allgemeinere Transition der Wert des Parameter `$transition` übergeben, und für die speziellere diejenige, die aus dem Event-Log rekonstruierte wurde. Das vorliegende Analysemodul wendet zur Berechnung der Transitionswahrscheinlichkeiten die in Abschnitt 2.1.1 beschriebenen Regeln an. Falls in den Modellen Konkretisierungen nach anderen Prinzipien durchgeführt wurden, kann für die Richtigkeit der Ergebnisse nicht garantiert werden.

Bei Konkretisierungen dürfen sowohl der Ausgangs- als auch der Zielzustand spezialisiert werden, d.h. durch einen Subzustand des ursprünglichen Zustands ersetzt werden. Des Weiteren kann das Ereignis, welches die Transition auslöst, mithilfe von Deskriptoren [Barnett et al., 2015, #events] genauer spezifiziert werden. Das Verstärken von Transitionsbedingungen ist ebenfalls erlaubt. Um diese zu vergleichen verwendet `compareTransitions` die Hilfsfunktion `compareConditions`, welche „true“ zurückliefert falls eine Bedingung mit der anderen im Sinne einer Spezialisierung übereinstimmt. Dazu prüft sie, ob die Teilzeichenkette, beginnend bei der Stelle null bis zur Länge der allgemeinen Bedingung, von der spezialisierten mit der allgemeinen Bedingung übereinstimmt und ob darauf die Zeichenkette „ `and` “ folgt. Hier ist anzumerken, dass durch Anhängen des Terms „`and false`“ eine Transitionsbedingung zwar formal korrekt spezialisiert werden kann, allerdings würde eine solche Bedingung niemals als „true“ evaluiert werden können, wodurch die Transition nicht mehr genommen werden kann. Semantisch entspricht das dem Weglassen von Transitionen, was bei Spezialisierungen nach dem Prinzip der *Observation Consistency* verboten ist. Da das vorliegende Analysemodul diesbezüglich keine inhaltliche Überprüfung der spezialisierten Transitionsbedingungen durchführt, obliegt diese Verantwortung den Benutzerinnen und Benutzern.

Die Funktion `getTransitionProbabilityForTargetState` in Quellcode 31 berechnet die absolute Wahrscheinlichkeit, mit der die Durchlaufzeit eines bestimmten Zustands gewichtet werden muss. Diese wird ermittelt, indem die Funktion den Prozess, ausgehend vom übergebenen Zustand, durch rekursive Aufrufe von hinten nach vorne durchgeht. Dafür benötigt sie fünf Parameter: das Prozessmodell (`$scxml`), den gewünschten Zustand (`$state`), die Parameter für die What-If-Analyse (`$changedTransitions` bzw. `$changedTransitionsFactors`) und die stark zusammenhängenden Komponenten (SZKs) des Prozessmodells (`$sccMap`). Der letzte Parameter wurde eingeführt um die Leistungsfähigkeit zu verbessern. Es wäre auch möglich gewesen, die SZKs erst in der Funktion `getTransitionProbabilityForTargetState` zu bestimmen, allerdings erfordert dies bei beinahe jedem Aufruf eine vollständige Tiefensuche (vgl. 3.2.4 Implementierung der Schleifenerkennung).

Zuerst prüft die Funktion, ob der übergebene Zustand (`$state`) der Initialzustand des gesamten Prozesses ist. Da ein SCXML-Modell nur einen Initialzustand besitzen darf, stellt dieser den Rekursionsanker dar und es wird die Wahrscheinlichkeit 100% zurückgegeben.

Im `else`-Zweig werden alle Transitionen auf den aktuellen Zustand bestimmt und mithilfe der SZKs (`$sccMap`) geprüft, ob er ein Wurzelelement einer Schleife ist. Trifft das nicht zu, ergibt sich die absolute Wahrscheinlichkeit aus der Summe der relativen Transitionswahrscheinlichkeiten multipliziert mit der absoluten Wahrscheinlichkeit der vorgehenden Zustände. Dazu iteriert die Funktion über die zuvor ermittelten Transitionen (`$transitions`), wobei für jede Transition das Produkt aus ihrer relativen Wahrscheinlichkeit (`getProbabilityFactor`) und dem Ergebnis des rekursiven Aufrufs von `getTransitionProbabilityFor-TargetState` mit ihrem Ausgangszustand für den Parameter `$state` gebildet wird.

Ist der übergebene Zustand das Wurzelelement einer Schleife, muss die Wahrscheinlichkeit entsprechend den Regeln der *Flow Analysis* bezüglich *rework*-Blöcken berechnet werden (vgl. 2.3.1 Quantitative Analyse). Es muss zwischen der Wahrscheinlichkeit, mit der Zustände in der Schleife zu gewichten sind und der, dass sie erreicht wird unterschieden werden. Ersteres wird von der Funktion `getProbabilityForRootNode` entsprechend der Formel `p = 1 / (1 - r)` [Dumas et al., 2013, S. 222] berechnet, wobei `r` die Wahrscheinlichkeit ist, dass die Schleife betreten wird. Der zweite Teil, zu sehen von Zeile 31 bis 41 in Quellcode 31, berechnet die Wahrscheinlichkeit, dass die Schleife erreicht wird, die anschließend mit der des Wurzelknotens multipliziert wird. Sollten mehrere Transitionen zum Wurzelelement der Schleife führen, werden deren (absolute) Wahrscheinlichkeiten aufsummiert.

Quellcode 31: Funktion getTransitionProbabilityForTargetState

```
1  declare function analysis:getTransitionProbabilityForTargetState(
2      $scxml as element(),
3      $state as element(),
4      $changedTransitions as element()*,
5      $changedTransitionsFactors as xs:decimal*,
6      $sccMap as map(*)
7  ) as xs:decimal {
8    if (analysis:stateIsInitialOfSCXML($state)) then
9      1
10   else
11     let $transitions := analysis:getTransitionsToState(
12       $scxml, $state, $includeSubstates, $checkParallel)
13     let $scc :=
14       analysis:getSCCForRootNode($scxml, $state, $sccMap)
15     return
16       if (empty($scc)) then
17         fn:sum(
18           for $transition in $transitions
19           return
20             analysis:getProbabilityFactor($transition,
21               $changedTransitions, $changedTransitionsFactors)
22             * analysis:getTransitionProbabilityForTargetState($scxml,
23               sc:getSourceState($transition), $changedTransitions,
24               $changedTransitionsFactors, $sccMap)
25           ,
26           0
27         )
28       else
29         analysis:getProbabilityForRootNode($scxml, $scc,
30           $changedTransitions, $changedTransitionsFactors)
31         * fn:sum(
32           for $transition in $transitions
33           return
34             if (not(functx:is-node-in-sequence(
35               sc:getSourceState($transition), $scc))) then
36               analysis:getProbabilityFactor($transition,
37                 $changedTransitions, $changedTransitionsFactors)
38               * analysis:getTransitionProbabilityForTargetState(
39                 $scxml, sc:getSourceState($transition),
40                 $changedTransitions, $changedTransitionsFactors,
41                 $sccMap)
42             else ()
43             ,
44             0
45         )
46 };
```

4 Qualitative Analyse

Dieses Kapitel erläutert die Funktionen des Analysemoduls für die qualitative Analyse von MBAs. Nach der Beschreibung der Benutzerschnittstelle werden die wichtigsten Aspekte der Umsetzung in XQuery beleuchtet.

4.1 Verwendung

Dieser Abschnitt beschreibt die Verwendung der Funktionen für die qualitative Analyse von MBAs. Die Schnittstellen der Funktionen sind in Tabelle 4 dargestellt. Mittels einer Fehler-Ursachen-Analyse (vgl. 2.3.2 Qualitative Analyse) werden problematische sowie zu Problemen führende Zustände im Prozessmodell eines MBAs identifiziert.

Die Funktion `getProblematicStates` liefert eine Sequenz von Zuständen aus dem SCXML-Modell zurück, bei denen die Durchlaufzeit mehr als einen konfigurierbaren, prozentualen Anteil der gesamten Durchlaufzeit beträgt. Dazu benötigt sie fünf Parameter. Die ersten beiden, `$mba` und `$level` beschreiben das zu untersuchende MBA und die Bezeichnung der gewünschten Ebene. Mithilfe des dritten, optionalen Parameters `$inState` wird die Menge der berücksichtigten MBAs eingeschränkt, so dass nur jene miteinbezogen werden, die sich in einem bestimmten Zustand befinden. Wird dieser Parameter nicht spezifiziert, wird diesbezüglich keine Einschränkung vorgenommen. Durch Verwendung des vierten Parameters `$excludeArchiveStates` können *Archival States*, d.h. durch das Attribut `mba:isArchiveState` gekennzeichnete Zustände, von der Analyse ausgeschlossen werden. Ein Weglassen dieses Parameters führt dazu, dass alle Zustände des Prozessmodells für die Auswertung verwendet werden. Der letzte Parameter `$threshold` ist vom Typ `xs:decimal` und legt fest, wie hoch der Anteil der Durchlaufzeit eines Zustands an der Gesamtzeit mindestens sein muss, damit dieser vom Analysemodul als problematisch deklariert wird.

Die zweite Funktion `getCausesOfProblematicStates` identifiziert die Ursachen von problematischen Zuständen eines Prozessmodells. Sie verlangt nach den gleichen Parametern wie `getProblematicStates`, d.h. `$mba`, `$level`, `$inState`, `$excludeArchiveStates` und `$threshold`. Dazu kommt der Parameter `$difference`, der die maximale Zeitspanne beschreibt die zwischen zwei Ereignissen vergehen darf, damit diese noch als voneinander abhängig gelten. Im Gegensatz zu `getProblematicStates` liefert die Funktion keine aus dem SCXML-Modell selektierten Zustände, sondern, in Anlehnung an *Why-Why*-Diagramme, eine Sequenz von verschachtelten `state`- oder `process`-Elementen zurück. Quellcode 32 zeigt ein Beispiel für den Rückgabewert der Funktion. Das untersuchte Prozessmodell besitzt den problematischen Zustand mit der Bezeichnung „A“. Dessen hohe Durchlaufzeit wird von den beiden Zuständen „C“ und

Tabelle 5: Schnittstelle qualitative Analyse

Funktion	**Parameter**	**Beschreibung**
getProblematicStates	$mba as element(), $level as xs:string, $inState as xs:string?, $excludeArchiveStates as xs:boolean?, $threshold as xs:decimal	Liefert alle Zustände, deren Durchlaufzeit einen bestimmten Anteil der gesamten Durchlaufzeit übersteigt.
getCausesOfProblematicStates	$mba as element(), $level as xs:string, $inState as xs:string?, $excludeArchiveStates as xs:boolean?, $threshold as xs:decimal, $difference as xs:dayTimeDuration	Identifiziert mögliche Ursachen für Probleme in Prozessen.

„F“ verursacht, der Zustand „C“ muss wiederum auf „D“ und dem Prozess auf der Ebene „l3“ warten. Alle diese Zustände sind problematisch, d.h. ihre Durchlaufzeit übersteigt den festgelegten Anteil an der Gesamtzeit. Alle `state`-Elemente ohne weitere Kindelemente stellen die fundamentalen Ursachen der Probleme des Prozessmodells bzw. der Konkretisierungshierarchie dar, da sie auf keine weiteren Ursachen zurückgeführt werden können. Ein `process`-Element zeigt an, dass zwar eine Abhängigkeit besteht, die Ursache aber nicht an einem einzelnen Zustand festgemacht werden kann.

Quellcode 32: Rückgabewert der Funktion getCausesOfProblematicStates

```
1 <state id="A">
2   <state id="C">
3     <state id="D" />
4     <process level="l3" />
5   </state>
6   <state id="F" />
7 </state>
```

4.2 Technische Dokumentation

Dieser Abschnitt beschreibt die XQuery-Umsetzung der Funktionen für die qualitative Analyse von MBAs. Dazu wird zuerst erläutert, wie problematische Zu-

stände in den Lebenszyklen von MBAs identifiziert werden und anschließend die Umsetzung der Fehler-Ursachen-Analyse geschildert.

4.2.1 Bestimmen von problematischen Zuständen

Die Funktion `getProblematicStates` bestimmt problematische Zustände im Prozess eines MBAs. Sie ist in Quellcode 33 abgebildet. Zuerst wird mithilfe der Funktion `getTotalCycleTime`, beschrieben in Abschnitt 3.1, die gesamte Durchlaufzeit des Prozesses berechnet. Die Durchlaufzeiten der Zustände und Transitionswahrscheinlichkeiten bleiben dabei unverändert, daher werden für die Parameter `$changedStates`, `$changedTransitions` und `$changedTransitionsFactors` jeweils leere Mengen übergeben. Um die maximal erlaubte Durchlaufzeit eines Zustands zu erhalten, wird die berechnete Gesamtzeit (`$totalCycleTime`) mit dem übergebenen Faktor (`$threshold`) multipliziert. Abschließend werden die Durchlaufzeiten aller Zustände des SCXML-Modells mit diesem Grenzwert verglichen. Ist sie größer, wird das entsprechende SCXML-Element des Zustands zurückgegeben.

4.2.2 Umsetzung der Fehler-Ursachen-Analyse

Die Funktion `getCausesOfProblematicStates` berechnet mithilfe von `getProblematicStates` alle problematischen Zustände des übergebenen MBAs. Anschließend ruft sie für jeden dieser Zustände die Hilfsfunktion `getCausesOfProblematicState` auf, die jeweils die zu dem Problem führenden Ursachen bestimmt. Quellcode 34 zeigt diese Funktion. Zur Identifikation der Fehlerquellen untersucht sie Subzustände, Synchronisationsbeziehungen und, basierend auf dem Wert des Parameters `$checkPrecedingStates`, vorhergehende Zustände.

Zu Beginn der Funktion muss geprüft werden, ob es sich bei dem zu untersuchenden Zustand um einen Initialzustand handelt. Beinhaltet der Parameter `$state` den Initialzustand eines Prozesses, ist das ursprüngliche Problem nicht auf einen einzelnen Zustand, sondern den gesamten Prozess zurückzuführen. Daher wird in diesem Fall ein `process`-Element erzeugt und zurückgegeben. Beinhaltet `$state` einen anderen Initialzustand, wird `getCausesOfProblematicState` für den übergeordneten Zustand erneut aufgerufen.

Für alle übrigen Zustände werden die zuvor genannten, möglichen Fehlerquellen untersucht. Die Funktion `getProblematicSubstates` prüft, ob sich ein Subzustand von `$state` in der Liste der problematischen Zustände (`$problematicStates`) befindet. Falls das der Fall ist, wird ein entsprechendes `state`-Element erzeugt sowie `getCausesOfProblematicState` für den problematischen Subzustand aufgerufen.

Quellcode 33: Funktion getProblematicStates

```
1  declare function analysis:getProblematicStates(
2      $mba as element(),
3      $level as xs:string,
4      $inState as xs:string?,
5      $excludeArchiveStates as xs:boolean?,
6      $threshold as xs:decimal
7  ) as element()* {
8    let $totalCycleTime :=
9      analysis:getTotalCycleTime(
10       $mba, $level, $inState, $excludeArchiveStates,
11       (), (), ()
12     )
13   let $cycleTimeThreshold := $totalCycleTime * $threshold
14
15   let $scxml := analysis:getSCXMLAtLevel($mba, $level)
16   let $states :=
17     analysis:getStates($scxml, $excludeArchiveStates)
18
19   return
20     for $state in $states
21     let $cycleTime :=
22       analysis:getCycleTimeForCompositeState(
23         $mba, $level, $inState, $state, (), (), ()
24       )
25     return
26       if ($cycleTime >= $cycleTimeThreshold) then
27         $state
28       else
29         ()
30 };
```

Quellcode 34: Funktion getCausesOfProblematicState

```
1  declare function analysis:getCausesOfProblematicState(
2      $mba as element(),
3      $level as xs:string,
4      $state as element(),
5      $inState as xs:string?,
6      $excludeArchiveStates as xs:boolean?,
7      $threshold as xs:decimal?,
8      $problematicStates as element()*,
9      $checkPrecedingStates as xs:boolean,
10     $difference as xs:dayTimeDuration?
11 ) as element()* {
12   if (analysis:stateIsInitialOfSCXML($state)) then
13     <process level="{$level}"/>
14   else if (analysis:stateIsInitial($state)) then
15     analysis:getCausesOfProblematicState($mba, $level, $state/..,
16       $inState, $excludeArchiveStates, $threshold,
17       $problematicStates, $checkPrecedingStates, $difference)
18   else
19     (
20       analysis:getProblematicSubstates($mba, $level, $state,
21         $inState, $excludeArchiveStates, $threshold,
22         $problematicStates, $difference)
23       ,
24       analysis:getProblematicSyncs($mba, $level, $state,
25         $excludeArchiveStates, $threshold, $difference)
26       ,
27       if ($checkPrecedingStates) then
28         analysis:getCauseOfProblematicSync($mba, $level, $state,
29           $excludeArchiveStates, $threshold, $difference)
30       else
31         ()
32     )
33 };
```

Abhängigkeiten über Synchronisationsbeziehungen werden von `getProblematicSyncs`, zu sehen in Quellcode 35, identifiziert. Sie untersucht Transitionsbedingungen (`cond`-Attribut) von Zustandsübergängen, die den übergebenen Zustand verlassen. Besitzt ein Zustandsübergang eine Bedingung mit einem Prädikat für Synchronisationsabhängigkeiten, wird diese von der Hilfsfunktion `getProblematicSyncsMBAAtLevelIsInState` untersucht. Diese benötigt unter anderem das verwendete Prädikat sowie seine Parameter. Beim Parsen der Vorbedingung muss allerdings aufgrund des Prädikats differenziert werden, da dieses entweder zwei (`$_everyDescendantAtLevelIsInState`, `$_someDescendantAtLevelIsInState` und `$_ancestorAtLevelIsInState`) oder drei (`$_isDescendantAtLevelInState` und `$_isAncestorAtLevelInState`) Parameter besitzen kann (vgl. 2.2.4 Prädikate für Synchronisationsabhängigkeiten). Die Hilfsfunktion `parseFunction` bestimmt das Prädikat, indem sie den Text vom Beginn der Zeichenkette der Vorbedingung bis zur ersten öffnenden Klammer selektiert. Bei denen mit zwei Parametern befinden sich die Bezeichnung der Ebene an erster Stelle und die des Zustands an zweiter. Die Ebene wird bei diesen Prädikaten von der Hilfsfunktion `parseLevelTwoParams` ermittelt und entspricht dem Teiltext der Vorbedingung zwischen der ersten öffnenden Klammer und dem ersten Beistrich danach. Die Bezeichnung des Zustands wird von `parseStateTwoParams` durch Selektion der Zeichenkette nach dem ersten Beistrich und vor der schließenden Klammer bestimmt. Dabei wird vorausgesetzt, dass beide Parameter, d.h. die Bezeichnung der Ebene und des Zustands, weder Beistriche noch Klammern enthalten. Bei den beiden Prädikaten mit drei Parametern beinhaltet der erste Parameter ein MBA, das als XPath-/XQuery-Ausdruck angegeben wird. Dabei muss davon ausgegangen werden, dass dieser sowohl Beistriche als auch Klammern enthält. Um dieses Prädikat entsprechend zu parsen, beginnt das Analysemodul am Ende der Zeichenkette. Der erste Parameter ergibt sich durch Selektion der Teilzeichenkette vor dem vorletzten Beistrich und nach der öffnenden Klammer des Synchronisationsprädikats (`parseObjectThreeParams`). Die Bezeichnung der Ebene als zweiter Parameter entspricht dem Teil zwischen dem letzten und vorletzten Beistrich (`parseLevelThreeParams`) und die Zustandsbezeichnung als letzter Parameter ist die Teilzeichenkette vor der letzten schließenden Klammer und nach dem letzten Beistrich (`parseStateThreeParams`). An dieser Stelle muss angemerkt werden, dass das vorliegende Analysemodul lediglich Vorbedingungen unterstützt, die aus genau einem der fünf in Abschnitt 2.2.4 beschriebenen Prädikate zur Synchronisation von Prozessmodellen bestehen. Um die Funktionalität des Moduls diesbezüglich auszuweiten müsste eine aufwendigere Logik zum Parsen der Transitionsbedingungen umgesetzt werden.

Quellcode 35: Funktion getProblematicSyncs

```
1  declare function analysis:getProblematicSyncs(
2      $mba as element(),
3      $level as xs:string,
4      $state as element(),
5      $excludeArchiveStates as xs:boolean?,
6      $threshold as xs:decimal?,
7      $difference as xs:dayTimeDuration?
8  ) as element()* {
9    for $t in analysis:getTransitionsLeavingState(
10     analysis:getSCXMLAtLevel($mba, $level), $state/@id)
11   return
12     if ($t/@cond) then
13       let $syncFunction := analysis:parseFunction($t/@cond)
14       return
15         if (($syncFunction = "$_everyDescendantAtLevelIsInState") or
16             ($syncFunction = "$_someDescendantAtLevelIsInState") or
17             ($syncFunction = "$_ancestorAtLevelIsInState")) then
18           analysis:getProblematicSyncsMBAAtLevelIsInState(
19             $mba, $level, $state,
20             $excludeArchiveStates, $threshold, $syncFunction,
21             analysis:parseLevelTwoParams($t/@cond),
22             analysis:parseStateTwoParams($t/@cond), (), $difference)
23         else if (($syncFunction = "$_isDescendantAtLevelInState") or
24             ($syncFunction = "$_isAncestorAtLevelInState"))then
25           analysis:getProblematicSyncsMBAAtLevelIsInState(
26             $mba, $level, $state,
27             $excludeArchiveStates, $threshold, $syncFunction,
28             analysis:parseLevelThreeParams($t/@cond),
29             analysis:parseStateThreeParams($t/@cond),
30             analysis:parseObjectThreeParams($t/@cond), $difference)
31         else
32           ()
33     else
34       ()
35 };
```

Die Prüfung von potentiell problematischen Synchronisationsbeziehungen wird von `getProblematicSyncsMBAAtLevelIsInState` durchgeführt, dazu koordiniert sie zwei Funktionsaufrufe. Die erste Funktion `isSyncCausingProblem`, dargestellt in Quellcode 36, stellt fest, ob die zu untersuchende Synchronisationsabhängigkeit Einfluss auf die Durchlaufzeit des Problemzustands `$state` hat. Trifft das zu, führt die zweite, `getCauseOfProblematicSync` (Quellcode 37), eine Untersuchung des in der Synchronisationsbeziehung angegebenen Zustands durch.

Um festzustellen, ob eine Synchronisationsbeziehung die Ursache eines Problems ist, muss, wie in Abschnitt 2.3.2 beschrieben, geprüft werden ob der problematische Zustand kurz nach dem Erfüllen der Vorbedingung verlassen wurde. Die ersten drei Parameter (`$mba`, `$level` und `$state`) beschreiben den problematischen Zustand, `$syncFunction` das verwendete Prädikat und die übrigen drei Parameter `$syncLevel`, `$syncStateId` und `$syncObj` legen den Zustand fest, auf den im Synchronisationsprädikat verwiesen wurde. Die Funktion iteriert über alle Konkretisierungen der Ebene `$level`, d.h. jene MBAs, die den problematischen Zustand `$state` beinhalten. Um zu ermitteln, wann dieser Zustand verlassen wurde, werden die jeweiligen State-Logs gebildet und alle entsprechenden Timestamps in die Variable `$untilProblemState` selektiert. In Prozessen mit Schleifen ist es möglich, dass ein Zustand öfter als einmal verlassen werden kann, daher enthält `$untilProblemState` unter Umständen eine Sequenz von Timestamps. Als nächstes müssen die Zeiten ermittelt werden, in denen die Vorbedingung der Transition erfüllt wurde (`$fromTimeSyncState`). Die Zeitpunkte unterscheiden sich dabei je nach verwendetem Prädikat. Eine Vorbedingung bestehend aus dem Prädikat `$_everyDescendantAtLevelIsInState` ist genau dann erfüllt, wenn die letzte Konkretisierung der angegebenen Ebene in den spezifizierten Zustand wechselt. Diese Funktionalität wird von der Hilfsfunktion `getMaxFromTimeOfState` umgesetzt. Analog dazu ist eine Vorbedingung `$_someDescendantAtLevelIsInState` dann erfüllt, wenn die erste Konkretisierung der angegebenen Ebene den definierten Zustand betritt, die Funktion `getMinFromTimeOfState` liefert diesen Zeitpunkt. Sowohl bei `getMaxFromTimeOfState` als auch `getMinFromTimeOfState` werden nur Konkretisierungen beachtet, die sich zum Zeitpunkt des Verlassens des Problemzustands in dem vom Prädikat spezifizierten Zustand befanden. Bei den drei übrigen Prädikaten `$_isDescendantAtLevelInState`, `$_isAncestorAtLevelInState` und `$_ancestorAtLevelIsInState` ist jeweils genau ein bestimmtes MBA betroffen. In diesen drei Fällen selektiert die Hilfsfunktion `getAllFromTimesOfState` alle Zeitpunkte aus dem State-Log der angegebenen Konkretisierung bzw. des Vorgängers, zu denen der in der Synchronisationsbeziehung festgelegte Zustand aktiv wurde. Hier ist es durch Schleifen erneut möglich, dass die Vorbedin-

gung mehrere Male erfüllt wurde und die Funktion deshalb eine Sequenz von Zeitpunkten zurückgibt. Am Ende von `isSyncCausingProblem` werden die Zeitpunkte des Verlassens des problematischen Zustands (`$untilProblemState`) mit den Zeitpunkten der Erfüllung der Vorbedingung (`$fromTimeSyncState`) verglichen. Enthält die Sequenz `$untilProblemState` zumindest einen Eintrag, der entweder einen gleichen oder höchstens um die in `$difference` festgelegte Dauer späteren Zeitpunkt aus `$fromTimeSyncState` beschreibt, kann davon ausgegangen werden, dass die Synchronisationsbeziehung das Verlassen des problematischen Zustands verzögerte. In diesem Fall liefert die Funktion `isSyncCausingProblem` den Wert „true“ zurück.

Quellcode 36: Funktion isSyncCausingProblem

```
1  declare function analysis:isSyncCausingProblem(
2      $mba as element(),
3      $level as xs:string,
4      $state as element(),
5      $syncFunction as xs:string,
6      $syncLevel as xs:string,
7      $syncStateId as xs:string,
8      $syncObj as xs:string?,
9      $difference as xs:dayTimeDuration?
10 ) as xs:boolean {
11   let $descendants :=
12     analysis:getDescendantsAtLevelOrMBA($mba, $level)
13   return
14     functx:is-value-in-sequence(
15       true()
16       ,
17       for $descendant in $descendants
18       let $stateLog := analysis:getStateLog($descendant)
19       let $untilProblemState :=
20         $stateLog/state[@ref = $state/@id]/@until
21       let $fromTimeSyncState :=
22         if ($syncFunction = "$_everyDescendantAtLevelIsInState"
23         ) then
24           analysis:getMaxFromTimeOfState($descendant, $syncLevel,
25             $syncStateId, $untilProblemState)
26         else if ($syncFunction = "$_someDescendantAtLevelIsInState"
27         ) then
28           analysis:getMinFromTimeOfState($descendant, $syncLevel,
29             $syncStateId, $untilProblemState)
30         else if ($syncFunction = "$_isDescendantAtLevelInState") then
31           analysis:getAllFromTimesOfState(xquery:eval($syncObj),
32             $syncStateId)
33         else if ($syncFunction = "$_ancestorAtLevelIsInState") then
34           analysis:getAllFromTimesOfState(
35             mba:getAncestorAtLevel($descendant, $syncLevel),
```

```
36              $syncLevel, $syncStateId)
37          else if ($syncFunction = "$_isAncestorAtLevelInState") then
38            analysis:getAllFromTimesOfState(xquery:eval($syncObj),
39              $syncStateId)
40          else ()
41        return
42          if (fn:empty($untilProblemState) or
43              fn:empty($fromTimeSyncState)) then
44            false()
45          else
46            for $untilTime in $untilProblemState
47            return
48              for $syncTime in $fromTimeSyncState
49              return analysis:timesAreSame($untilTime, $syncTime,
50              $difference)
51      )
52  };
```

Die Funktion `getCauseOfProblematicSync` wird aufgerufen wenn festgestellt wurde, dass das späte Erreichen des Zustands (`$state`) zu Verzögerungen in einem über eine Synchronisationsbeziehung verknüpften Prozess führt. Sie muss ermitteln, warum dieser Zustand so spät aktiv wurde. Eine Ursache dafür kann ein vorhergehender problematischer Zustand sein. Um Probleme zu identifizieren, werden mithilfe der Funktion `getProblematicStates` zuerst die problematischen Zustände des Prozesses ermittelt (`$problematicStates`). Anschließend werden jene Zustände in die Variable `$precedingStates` gespeichert, die direkte Vorgänger des Zustands `$state` sind. Alle Vorgänger, die sich ebenfalls in der Liste der problematischen Zustände befinden, sind Ursachen dafür, dass der Zustand `$state` verspätet erreicht wurde. Deshalb wird für jeden dieser Vorgänger ein `state`-Element erzeugt, des Weiteren folgt jeweils ein Aufruf der zuvor beschrieben Funktion `getCausesOfProblematicState`, da jeder dieser Problemzustände auf seine Ursachen untersucht werden muss. Ist kein Vorgänger von `$state` ein Problemzustand, müssen deren Vorgänger weiter untersucht werden, bis entweder der Anfangszustand des Prozesses oder ein problematischer Zustand gefunden wurden. Das geschieht ebenfalls über den Aufruf von `getCausesOfProblematicState`, allerdings wird der letzte Parameter (`$checkPrecedingStates`) auf „true“ gesetzt. Das führt dazu, dass neben den Überprüfungen von Subzuständen und weiteren Synchronisationsbeziehungen auch vorhergehende Zustände untersucht werden (siehe Zeile 27-29 in Quellcode 34).

Quellcode 37: Funktion getCauseOfProblematicSync

```
1  declare function analysis:getCauseOfProblematicSync(
2      $mba as element(),
3      $level as xs:string,
4      $state as element(),
5      $excludeArchiveStates as xs:boolean?,
6      $threshold as xs:decimal?,
7      $difference as xs:dayTimeDuration?
8  ) as element()* {
9    let $scxml := analysis:getSCXMLAtLevel($mba, $level)
10   let $problematicStates :=
11     analysis:getProblematicStates($mba, $level, (),
12       $excludeArchiveStates, $threshold)
13   let $precedingStates :=
14     analysis:getTransitionsToState($scxml, $state)/..
15   return
16     if ($precedingStates/@id = $problematicStates/@id) then
17       for $prec in $precedingStates[@id = $problematicStates/@id]
18       return
19         <state id="{fn:string($prec/@id)}">
20           {
21             analysis:getCausesOfProblematicState(
22               $mba, $level, $prec, (), $excludeArchiveStates,
23               $threshold, $problematicStates, false(), $difference)
24           }
25         </state>
26     else
27       functx:distinct-deep(
28         for $prec in $precedingStates
29         return
30           analysis:getCausesOfProblematicState(
31             $mba, $level, $prec, (), $excludeArchiveStates,
32             $threshold, $problematicStates, true(), $difference)
33       )
34 };
```

5 Fazit und Ausblick

Diese Arbeit stellt ein Analysemodul zur Auswertung von mehrstufigen Geschäftsprozessen vor. Dazu bedient sie sich dem von Schütz vorgestelltem Konzept von *Multilevel Business Artifacts* (MBAs) [Schuetz, 2015]. Es erlaubt die Darstellung der Geschäftsprozesse eines Unternehmens, gegliedert in hierarchische Ebenen. Die Möglichkeit, Prozessmodelle im Zuge von Konkretisierungen zu spezialisieren, begünstigt dabei die flexible Modellierung von Abläufen [Schütz et al., 2012]. Da für die Abbildung der Prozessmodelle der SCXML-Standard [Barnett et al., 2015] verwendet wird, können diese semi-automatisiert von einem SCXML-Interpreter ausgeführt werden. Dies erzeugt Logs, die vom vorgestellten Modul für die Analyse verwendet werden.

Das Analysemodul ermittelt nach dem Prinzip der *Flow Analysis* [Dumas et al., 2013, S. 213 ff.] die Durchlaufzeiten von Prozessen und Zuständen (quantitative Analyse). Die Berechnung erfolgt dabei immer auf einer festgelegten Ebene eines MBAs. Ist die spezifizierte Ebene nicht der Top-Level, liefert das Modul aggregierte Werte, d.h. durchschnittliche Durchlaufzeiten über alle entsprechenden Konkretisierungen. Hier können Analystinnen und Analysten von der heterohomogenen Eigenschaft von MBAs profitieren. Durch die Navigation entlang der Konkretisierungshierarchie von einem MBA zu seinen Konkretisierungen kann der Detaillierungsgrad erhöht werden, um bestimmte Prozesse genauer zu betrachten. Umgekehrt erlaubt die Selektion des übergeordneten MBAs die Berechnung von allgemeineren Werten [Schuetz, 2015, S. 197 ff.]. Neben der quantitativen Analyse bietet das Modul auch Funktionen zur Identifikation von Schwachstellen im Prozess an (qualitative Analyse) [Dumas et al., 2013, S. 185 ff.].

Es wurden einige Aspekte angemerkt, die den Ausgangspunkt für zukünftige Arbeiten darstellen können. So ist es denkbar, das Analysemodul zu erweitern, damit neben der Durchschnittsberechnung alternative Aggregationsfunktionen verwendet werden können, z.B. Median, Minimum oder Maximum. Darüber hinaus können die vorgestellten Techniken der *Flow Analyse* in leicht abgewandelter Form zur Berechnung von anderen Metriken, beispielsweise der Kosten, verwendet werden (siehe auch [Dumas et al., 2013, S. 227 ff.]). Dies setzt allerdings voraus, dass MBAs über die entsprechenden Informationen verfügen, beispielsweise die von einem Zustand verursachten Kosten.

Um die qualitative Analyse weiter zu verbessern kann die Auswertung von Synchronisationsbeziehungen ausgebaut werden. Dazu muss das Parsen der Transitionsbedingungen erweitert werden, so dass die Verarbeitung von mehreren Prädikaten oder die Spezialisierung von Synchronisationsbeziehungen ermöglicht wird. Darüber hinaus bietet sich die Umsetzung weiterer, der von Schütz [Schuetz, 2015, S. 76 ff.] vorgestellten, Prädikate zur Synchronisation von MBAs an.

6 Literatur

[Barnett et al., 2015] Barnett, J., Akolkar, R., Auburn, R., Bodell, M., Burnett, D. C., Carter, J., McGlashan, S., Lager, T., Helbing, M., Hosn, R., Raman, T., Reifenrath, K., Rosenthal, N., und Roxendal, J. (2015). State Chart XML (SCXML): State Machine Notation for Control Abstraction W3C Recommendation 1 September 2015. Abgerufen am 16.08.2016 von https://www.w3.org/TR/2015/REC-scxml-20150901/.

[BaseX, 2016] BaseX (2016). Basex Documentation, Version 8.5. Abgerufen am 05.08.2016 von http://files.basex.org/releases/8.5/BaseX85.pdf.

[Becker et al., 2009] Becker, J., Mathas, C., und Winkelmann, A. (2009). *Bedeutung des Geschäftsprozessmanagements*, Seiten 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Clark und DeRose, 2015] Clark, J. und DeRose, S. (2015). XML Path Language (XPath). Abgerufen am 15.09.2016 von https://www.w3.org/TR/xpath/.

[Domschke und Drexl, 2005] Domschke, W. und Drexl, A. (2005). *Graphentheorie*, Seiten 65–80. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Dumas et al., 2013] Dumas, M., Rosa, M. L., Mendling, J., und Reijers, H. A. (2013). *Fundamentals of Business Process Management*. Springer Publishing Company, Incorporated.

[Farkisch, 2011] Farkisch, K. (2011). *Datenanalyse, Navigation anhand multidimensionaler Funktionen und Operatoren*, Seiten 39–44. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Gadatsch, 2010] Gadatsch, A. (2010). *Grundkurs Geschäftsprozess-Management: Methoden und Werkzeuge für die IT-Praxis: Eine Einführung für Studenten und Praktiker*. Vieweg+Teubner, Wiesbaden.

[Günther und Verbeek, 2014] Günther, C. W. und Verbeek, E. (2014). XES-Standard Definition (2014). Abgerufen am 05.08.2016 von http://www.xes-standard.org/_media/xes/xesstandarddefinition-2.0.pdf.

[Hammer, 2010] Hammer, M. (2010). *What is Business Process Management?*, Seiten 3–16. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Mintzberg, 1979] Mintzberg, H. (1979). *The Structuring of Organizations: A Synthesis of the Research*. Theory of Management Policy Series. Prentice-Hall.

[Neumayr et al., 2010] Neumayr, B., Schrefl, M., und Thalheim, B. (2010). Hetero-Homogeneous Hierarchies in Data Warehouses. In *Proceedings of the Seventh Asia-Pacific Conference on Conceptual Modelling - Volume 110*, APCCM '10, Seiten 61–70, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

[OMG, 2011] OMG (2011). Superstructure, Version 2.4.1. Abgerufen am 05.08.2016 von http://www.omg.org/spec/UML/2.4.1/.

[Schrefl und Stumptner, 2000] Schrefl, M. und Stumptner, M. (2000). *Behavior Consistent Inheritance in UML*, Seiten 527–542. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Schrefl und Stumptner, 2002] Schrefl, M. und Stumptner, M. (2002). Behavior-Consistent Specialization of Object Life Cycles. *ACM Trans. Softw. Eng. Methodol.*, 11(1):92–148.

[Schuetz et al., 2015] Schuetz, C. G., Neumayr, B., und Schrefl, M. (2015). Multilevel Modeling for Business Process Automation. In *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*, Seiten 51–60.

[Schuetz, 2015] Schuetz, G. C. (2015). *Multilevel Business Processes: Modeling and Data Analysis*. Springer Fachmedien Wiesbaden, Wiesbaden.

[Schütz et al., 2013] Schütz, C., Delcambre, L. M. L., und Schrefl, M. (2013). *Multilevel Business Artifacts*, Seiten 304–315. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Schütz et al., 2012] Schütz, C., Schrefl, M., und Delcambre, L. M. (2012). Multilevel Business Process Modeling: Motivation, Approach, Design Issues, and Applications. In *Proceedings of the 5th Ph. D. Workshop on Information and Knowledge*, Seiten 91–94. ACM.

[Tarjan, 1972] Tarjan, R. (1972). Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*.

[Tomić und Spasojević Brkić, 2011] Tomić, B. und Spasojević Brkić, V. (2011). Effective Root Cause Analysis and Corrective Action Process. *Journal of Engineering Management and Competitiveness (JEMC)*, 1(1/2):16–20.

[Turau, 2009] Turau, V. (2009). *Algorithmische Graphentheorie*. De Gruyter Studium. Oldenbourg.

[W3C, 2008] W3C (2008). Extensible Markup Language (XML) 1.0 (Fifth Edition). Abgerufen am 29.10.2016 von http://www.w3.org/TR/2008/REC-xml-20081126/.

[W3C, 2014a] W3C (2014a). XPath and XQuery Functions and Operators 3.0 – W3C Recommendation 8 April 2014. Abgerufen am 17.08.2016 von https://www.w3.org/TR/2014/REC-xpath-functions-30-20140408/.

[W3C, 2014b] W3C (2014b). XQuery 3.0: An XML Query Language. Abgerufen am 10.09.2016 von https://www.w3.org/TR/2014/REC-xquery-30-20140408/.

[W3C, 2015] W3C (2015). XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition). Abgerufen am 16.09.2016 von https://www.w3.org/TR/2010/REC-xpath-datamodel-20101214/.

[Walther, 1978] Walther, H. (1978). *Anwendungen der Graphentheorie*. Vieweg+Teubner Verlag, Wiesbaden.

[zur Mühlen und Shapiro, 2010] zur Mühlen, M. und Shapiro, R. (2010). *Business Process Analytics*, Seiten 137–157. Springer Berlin Heidelberg, Berlin, Heidelberg.