ARTHUR VETTER

AUTOMATISIERTE ERKENNUNG VON FEHLERN BEI DER WARTUNG VON IT-SERVICES

KIT Scientific Publishing

Arthur Vetter

Automatisierte Erkennung von Fehlern bei der Wartung von IT-Services

Automatisierte Erkennung von Fehlern bei der Wartung von IT-Services

von
Arthur Vetter

KIT Scientific Publishing

Dissertation, Karlsruher Institut für Technologie
KIT-Fakultät für Wirtschaftswissenschaften

Tag der mündlichen Prüfung: 14. März 2019
Gutachter: Prof. Dr. Andreas Oberweis, Prof. Dr. Alexander Mädche

Impressum

Karlsruher Institut für Technologie (KIT)
KIT Scientific Publishing
Straße am Forum 2
D-76131 Karlsruhe

www.ksp.kit.edu

Print on Demand 2019 – Gedruckt auf FSC-zertifiziertem Papier

ISBN 978-3-7315-0921-9
DOI 10.5445/KSP/1000092728

Automatisierte Erkennung von Fehlern bei der Wartung von IT-Services

zur Erlangung des akademischen Grades eines

Doktors der Wirtschaftswissenschaften

von der KIT-Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Arthur Vetter

aus Stuttgart

Tag der mündlichen Prüfung: 14.03.2019
Erster Gutachter: Prof. Dr. Andreas Oberweis
Zweiter Gutachter: Prof. Dr. Alexander Mädche

Vorwort

Diese Arbeit entstand während meiner Zeit als Doktorand am Institut für Angewandte Informatik und Formale Beschreibungsverfahren des Karlsruher Instituts für Technologie (KIT) parallel zu meinen Aufgaben als Principal Consultant bei der PROMATIS software GmbH in Ettlingen.

Ganz herzlich bedanke ich mich bei Herrn Prof. Dr. Andreas Oberweis für die Betreuung meines Promotionsvorhabens, sowie seinen kritischen Anmerkungen und konstruktiven Ratschlägen. Ebenso danke ich Herrn Prof. Dr. Alexander Mädche für die Übernahme des Korreferats. Herrn Prof. Dr. Ali Sunyaev und Herrn Prof. Dr. J. Philipp Reiss danke ich für die Diskussion der Arbeitsergebnisse und dass sie sich als Mitglieder des Prüfungskolloquiums zur Verfügung gestellt haben.

Besonders bedanke ich mich bei Herrn Dr. Frank Schönthaler, dem Vorsitzenden der Geschäftsleitung der PROMATIS software GmbH, für die uneingeschränkte Unterstützung meines Promotionsvorhabens während meiner beruflichen Tätigkeit. Außerdem danke ich ganz herzlich meinem Arbeitskollegen Dr. Thomas Karle für den Erfahrungs- und Gedankenaustausch sowie seine stets aufmunternden Worte.

Weiterhin danke ich insbesondere Dr. Andreas Schoknecht für die vielen Diskussionen und das Korrekturlesen der formalen Abschnitte. Des Weiteren danke ich den folgenden Kollegen und Kolleginnen des Instituts AIFB und der PROMATIS software GmbH für die stets gute und erfolgreiche Zusammenarbeit: PD Dr. Agnes Koschmider, Andreas Drescher, Meike Ullrich, Andreas Fritsch, Hannah Chipaux, Johannes Michler und Mana Taghdiri.

Mein liebster Dank gilt meiner Partnerin Tatjana Anita Zehnle, die sich geduldig meine Ideen angehört hat und mein Vorhaben immer uneingeschränkt unterstützt hat. Außerdem hat sie das Korrekturlesen des Manuskripts übernommen und wertvolle Hinweise zur Verbesserung meiner wissenschaftlichen Vorträge gegeben. Zuletzt möchte ich meiner Familie danken, die mich beim Erreichen meiner Ziele und meinem persönlichen Weg dorthin immer gestärkt hat.

Stuttgart, März 2019

Arthur Vetter

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Abkürzungsverzeichnis

BPMN	Business Process Model and Notation
CCTA	Central Computer and Telecommunications Agency
CEP	Complex Event Processing
CI	Configuration Item
CMDB	Configuration Management Database
CMS	Configuration Management System
DMTF	Distributed Management Task Force, Inc.
EPK	Ereignisgesteurte Prozessketten
EPL	Event Processing Language
IEC	International Electrotechnical Commission
ISO	International Organization for Standards
IT	Informationstechnologie
ITIL	IT Infrastructure Library
ITSM	IT-Service-Management
MOF	Microsoft Operations Framework
OGC	Office of Government Commerce
RFC	Request for Change
SLA	Service Level Agreement
SLM	Service Level Management
SOM	Semantisches Objektmodell
GEMS	Generic Error Modeling System

TOSCA	Topology and Orchestration Specification for Cloud Applications
UML	Unified Modeling Language
W3C	World Wide Web Consortium
XQuery	XML Query
XManiLa	XML Manipulation Language
XML	Extensible Markup Language

Symbolverzeichnis

$\in$	Element von
$\notin$	Kein Element von
γ	Gruppierung
π	Projektion
ρ	Umbenennung
σ	Selektion
$\succ$	Nachfolgend
$\bowtie$	Relationaler Verbund
$\times$	Kartesisches Produkt
$\emptyset$	Leere Menge
$\cdot x$	Vorbereich von Knoten x
$x \cdot$	Nachbereich von Knoten x
$>$	Größer
$\geq$	Größer oder gleich
$<$	Kleiner
$\leq$	Kleiner oder gleich
$\wedge$	Und
$\vee$	Oder

1 Einleitung und Problemstellung

Der Einsatz von Informationstechnologien nimmt in Unternehmen seit Jahren stetig zu und hat für viele Unternehmen eine geschäftskritische Bedeutung erreicht [MaHG15]. Es ist davon auszugehen, dass aktuelle Trends, wie Digitalisierung, Künstliche Intelligenz, Cloud Computing und Industrie 4.0 die Bedeutung der IT für Unternehmen noch weiter erhöhen werden. Dies führt dazu, dass ein Ausfall von IT-Services, mit dem wichtige Geschäftsprozesse unterstützt werden, zu massiven negativen Auswirkungen für Unternehmen führen kann, wie folgendes Beispiel verdeutlicht:

Am 21. April 2011 kam es in einem Rechenzentrum von Amazon zu einer Störung, die zu einem temporären Netzwerkausfall führte, wodurch die angebotenen IT-Services von Amazon nicht mehr erreicht werden konnten. Die Störung hatte einen temporären und sogar dauerhaften Verlust von Kundendaten zur Folge. Grund der Störung war die fehlerhafte Durchführung eines geplanten Netzwerk-Upgrades. Für die Durchführung des Upgrades sollte der Datenverkehr über einen redundanten Router geleitet werden, jedoch war der Router nicht für die Datenlast ausgelegt, so dass es zu Netzwerkstörungen kam [Amaz11].

Um Änderungen von IT-Services vorzubereiten und kontrolliert durchzuführen, wird der Einsatz eines IT-Change-Managements empfohlen [Scot99, BoJK08, Offi11a]. Der De-facto-Standard im IT-Service-Management, die IT Infrastructure Library (ITIL), beschreibt die einzelnen Aktivitäten, die im Rahmen eines Change Managements durchzuführen sind, um eine IT-Service-Wartung kontrolliert durchzuführen. Eine Maßnahme von Amazon, um derartige Fehler wie oben beschrieben zu vermeiden, war die Überprüfung und Verbesserung des Change Management Prozesses bei Amazon [Amaz11]. Verschiedene Studien bestätigen die Bedeutung einer kontrollierten Durchführung von IT-Service-Wartungen, da Fehler während einer IT-Service-Wartung eine der häufigsten Gründe für IT-Service-Ausfälle sind:

In einer Studie von 1986 berichtet Gray [Gray86], dass der Hauptgrund für Systemausfälle sogenannte „System Management“ Fehler sind. Zu dieser Kategorie zählt er Fehler beim Deployment, fehlerhafte Systemwartungen und Systemkonfigurationen.

Oppenheimer et al. [OpGP03] haben die Gründe für Internet Service Ausfälle untersucht. Bei den untersuchten Internet Services sind Durchführungsfehler für 18-36% der Ausfälle verantwortlich. Hardwarefehler sind für 1-18% der Ausfälle und Softwarefehler für 14-59%

der Ausfälle verantwortlich. Bei den restlichen untersuchten Ausfällen konnte nicht identifiziert werden, was die Gründe für die Ausfälle waren.

Eine Untersuchung von Cloud Computing-Anwendungssystemen kommt zu dem Schluss, dass Durchführungsfehler mit einem Auftreten von 14% die vierthäufigste Ursache für IT-Service-Ausfälle sind [GHLP14].

Pertet und Narasimhan [PeNa05] haben über 40 tatsächlich aufgetretene Störungen von IT-Services auf ihre Fehlerursache überprüft und diese in vier Kategorien eingeteilt. Eine Kategorie sind Durchführungsfehler. Diese werden unterteilt in Prozessfehler, z.B. weil das Starten eines Web Servers vergessen wurde, Konfigurationsfehler und sonstige Fehler, z.B. weil Hardware aus Versehen fallen gelassen wurde oder jemand über ein Stromkabel stolperte und dieses daraufhin aus der Steckdose gezogen wurde.

Gemäß einer Studie von Elliot, in der über 20 Fortune 1000 Unternehmen befragt wurden, wird erwartet, dass die durchschnittliche Anzahl an Software-Deployments weiter zunehmen wird [Elli14], wodurch mit einer weiteren Zunahme von Service-Ausfällen aufgrund von Durchführungsfehlern zu rechnen ist, wenn diese nicht abgestellt werden können.

Trotz der Bedeutung von Durchführungsfehlern, bemängeln Oppenheimer et al. [OpGP03], wurden kaum Anstrengungen unternommen, um Methoden und Tools zur Überwachung und Kontrolle von Durchführungsfehlern zu entwickeln.

1.1 Ziele der Arbeit

Ausgehend von der Problemstellung ist das Ziel dieser Arbeit die Entwicklung einer generischen Methode zur Kontrolle der Durchführung von IT-Service-Wartungen, um Ausfälle von IT-Services aufgrund von Wartungsfehlern zu vermeiden. Zur weiteren Erläuterung der Ziele dieser Arbeit wird der Begriff der Untersuchungssituation verwendet [Fers79]. Demnach wird eine Untersuchungssituation aus dem Tripel *Untersuchungsobjekt* bzw. *Untersuchungsgegenstand*, *Untersuchungsziel* und *Untersuchungsverfahren* bestimmt. Ein *Untersuchungsgegenstand* wird mittels seiner bekannten Eigenschaften beschrieben. Das *Untersuchungsziel* definiert, welche bisher unbekannten Eigenschaften des *Untersuchungsgegenstands* untersucht werden. Das *Untersuchungsverfahren* beschreibt, welche Methoden angewendet werden, um das *Untersuchungsziel* zu erreichen. Kann mithilfe des angewandten *Untersuchungsverfahrens* das *Untersuchungsziel* erreicht werden, wird von einer Problemlösung gesprochen.

1.1.1 Untersuchungsgegenstand

Der Untersuchungsgegenstand dieser Arbeit ist die Wartung von IT-Services. Wie in der Problemstellung erläutert, ist ein großer Anteil aller Serviceausfälle auf Fehler während der Durchführung einer IT-Service-Wartung zurückzuführen. Eine typische IT-Service-Wartung ist zum Beispiel die Installation eines Softwareupdates für eine Anwendung oder die Anpassung von Konfigurationsparametern.

1.1.2 Untersuchungsziel

Das Untersuchungsziel dieser Arbeit ist die Entwicklung einer Methode, mit der Fehler bei der Durchführung von IT-Service-Wartungen online während der Durchführung erkannt werden können. Dadurch sollen Wartungsfehler noch während der Wartung behoben werden können und IT-Service-Ausfälle aufgrund von Wartungsfehlern vermieden werden. Durchführungsfehler lassen sich in Konfigurationsfehler und Ablauffehler unterscheiden [PeNa05]. In dieser Arbeit werden beide Fehlertypen berücksichtigt.

1.1.3 Untersuchungsverfahren

Zur Erreichung des Untersuchungsziels wird eine Methode zur Analyse von Abweichungen zwischen dem geplanten Ablauf einer IT-Service-Wartung und der tatsächlich durchgeführten IT-Service-Wartung entwickelt. Dazu soll der Einsatz von Anti-Pattern überprüft werden, wie diese bereits im Compliance-Umfeld angewendet werden [ABEE15]. Dabei wird davon ausgegangen, dass der in der Design-Phase erstellte Wartungsplan den Soll-Ablauf zur Durchführung der IT-Service-Wartung beschreibt. Mithilfe von Anti-Pattern sollen während der eigentlichen Durchführung der IT-Service-Wartung Abweichungen vom Wartungsplan erkannt werden. Der Nutzen dieser Methode liegt zum einen in der Kontrolle der Wartungsdurchführung, so dass Fehler bereits während der IT-Service-Wartung erkannt und beseitigt werden können, bevor ein IT-Service wieder produktiv genommen wird. Zum anderen bietet die Anwendung von Anti-Pattern den Vorteil, dass bei eintretenden Fehlern keine zusätzlichen Kosten für eine Root-Cause-Analyse notwendig sind [LMMR13] und nicht in den einzelnen Anwendungssystemen nach Konfigurationsfehlern gesucht werden muss, sondern eine Analyse auf Basis des Wartungsplans erfolgen kann. Für die Darstellung des Wartungsplans und der Ableitung der Anti-Pattern soll auf Methoden zur Modellierung von Geschäftsprozessen zurückgegriffen werden. Das genaue Untersuchungsverfahren wird anhand folgender drei Fragestellungen beschrieben:

1. *Wie können IT-Service-Wartungen modelliert werden?*

   Damit die Durchführung einer IT-Service-Wartung gegen ein Modell der Wartungsdurchführung geprüft werden kann, werden zunächst die Anforderungen an eine Modellierungssprache zur Beschreibung von IT-Service-Wartungen ermittelt. Hierfür wird mittels einer Literaturrecherche geprüft, welche Fehler bei der Durchführung einer IT-Service-Wartungen auftreten können, um daraus die Anforderungen an eine Modellierungssprache abzuleiten.

2. *Welche Anti-Pattern eignen sich zur Entdeckung von Fehlern bei der Durchführung von IT-Service-Wartungen?*

   Auf Basis der zuvor mittels Literaturrecherche ermittelten typischen Fehler, die bei der Durchführung einer IT-Service-Wartung auftreten können, werden Anti-Pattern definiert, mit denen diese Fehler während der Durchführung einer IT-Service-Wartungen erkannt werden können. Zusätzlich wird beschrieben, wie mithilfe der Anti-Pattern und einem Modell der IT-Service-Wartung die Fehlerursache ermittelt werden kann.

3. *Wie kann die Überwachung der Anti-Pattern während der Wartungsdurchführung erfolgen?*

   Es wird eine Architektur entwickelt, prototypisch umgesetzt und evaluiert, ob mit den entwickelten Anti-Pattern Fehler online während der Durchführung einer IT-Service-Wartung entdeckt werden können. Zur Auswertung der Anti-Pattern wird eine Complex Event Processing Engine eingesetzt, die kontinuierlich Log-Dateien und andere Metriken von IT-Service-Komponenten in einem Datenstrom analysiert.

Bei der Anwendung des Untersuchungsverfahrens werden folgende Annahmen getroffen:

- Alle Änderungen an einem IT-Service werden in Logs dokumentiert, d.h. wenn beispielsweise eine Änderung an einer Datenbank durchgeführt wird, wird diese Änderung in eine Log-Datei geschrieben.
- Konfigurationen werden nicht fest im Code einer Anwendung vorgenommen, sondern in separate Konfigurationsdateien geschrieben.
- Die Personen, die eine IT-Service-Wartung planen, haben vollständiges Wissen über die durchzuführenden Aktivitäten der IT-Service-Wartung, d.h. sie wissen, welche Aktivitäten wie, auf welcher IT-Service-Komponente, und in welcher Reihenfolge durchzuführen sind.

- IT-Service-Wartungen werden nur nach vorgehergehender Genehmigung durchgeführt, d.h. alle durchgeführten und durchzuführenden IT-Service-Wartungen sind bekannt.

1.2 Aktueller Forschungsstand

Arbeiten zur Entdeckung von Fehlern bei IT-Service-Wartungen, bzw. zur Vermeidung von Fehlern bei der Durchführung von IT-Service-Wartungen können in drei Gruppen strukturiert werden:

- Arbeiten, die vor der eigentlichen IT-Service-Wartung angesiedelt sind, und durch eine korrekte Planung und durch automatisierte Konfigurationsprüfungen versuchen, Durchführungsfehler zu verhindern. Die Limitation dieser Arbeiten ist jedoch, dass bei der eigentlichen Durchführung einer IT-Service-Wartung, trotz guter und korrekter Vorbereitung, Fehler nicht verhindert werden können. Arbeiten, die dieser Kategorie zugeordnet werden können, werden in Kapitel 1.2.1 vorgestellt.
- Arbeiten, die explizit die Durchführung von IT-Service-Wartungen untersuchen und Durchführungsfehler verhindern wollen, indem sie die Durchführung einer IT-Service-Wartung überwachen oder die Durchführung automatisieren. Die Vor- und Nachteile dieser Arbeiten werden in Kapitel 1.2.2 beschrieben.
- Arbeiten, die sich mit der Aufdeckung von Fehlern durch IT-Service-Wartungen beschäftigen, jedoch diese nach der Durchführung einer IT-Service-Wartung und nicht währenddessen aufdecken. Der Nachteil dieser Ansätze ist, dass sich ein Durchführungsfehler bis zu seiner Erkennung bereits im IT-Service manifestiert hat und somit bereits Schaden anrichten kann. Da diese Ansätze nur entfernt verwandt zu dieser Arbeit sind, werden sie nicht weiter beschrieben.

Abbildung 1.1 stellt die Gruppen verwandter Forschungsarbeiten anhand des zeitlichen Ablaufs einer IT-Service-Wartung dar.

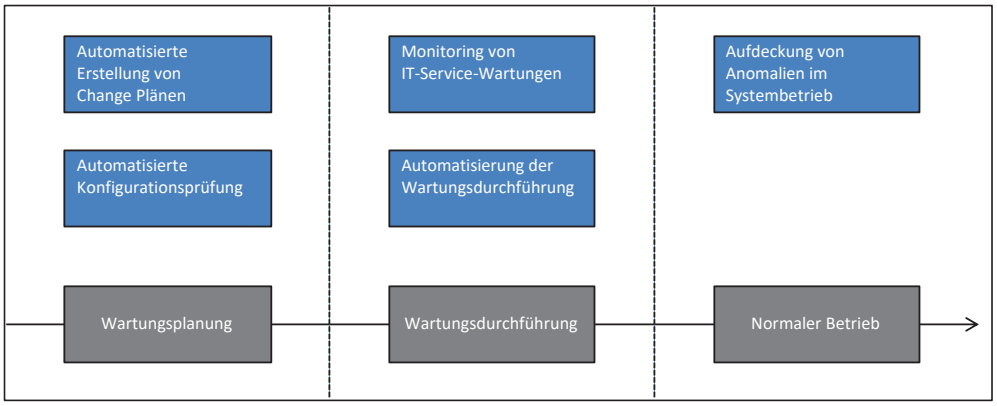


Abbildung 1.1: Strukturierung des Forschungsstands

1.2.1 Vor der Wartungsdurchführung

Arbeiten, in denen untersucht wird, wie sich Durchführungsfehler bereits im Voraus vermeiden lassen, werden nachfolgend in zwei Kategorien unterteilt.

Zunächst werden Arbeiten vorgestellt, in denen die automatische Erstellung von sogenannten „Change Plänen“ untersucht wird. Ein Change Plan ist ein zentrales Werkzeug des Change Managements. Im Change Plan werden Aktivitäten, Verantwortlichkeiten und weitere Informationen zur Durchführung einer IT-Service-Wartung beschrieben. Mit diesen Change Plänen wird bei der Planung einer IT-Service-Wartung sichergestellt, dass der geplante Ablauf der IT-Service-Wartung korrekt ist, es zu keinen Wechselwirkungen mit anderen IT-Service-Wartungen kommt, sowie keine Service-Beeinträchtigungen aufgrund der Art der Durchführung zu erwarten sind.

Zusätzlich werden Arbeiten vorgestellt, in denen beschrieben wird, wie sich Konfigurationen a priori auf Korrektheit überprüfen lassen, so dass vermieden werden kann, dass eine IT-Service-Wartung eventuell korrekt durchgeführt wurde, es jedoch aufgrund der Konfiguration von IT-Service-Komponenten zu Servicebeeinträchtigungen kommen kann.

Automatiserte Erstellung von Change Plänen

Arbeiten zur automatisierten Erstellung von Change Plänen ermöglichen die Vorbeugung von Ablauffehlern während der Durchführung einer IT-Service-Wartung, da diese Abhängigkeiten zwischen verschiedenen IT-Service-Komponenten und anderen IT-Service-Wartungen aufdecken und versuchen, diese Abhängigkeiten aufzulösen.

Eine der ersten Arbeiten zur automatischen Erstellung von Change Plänen und der Realisierung in einem Prototypen ("CHAMPS") stammt von Keller et al. [KHWW04] und kann als Startschuss für darauf folgende Arbeiten zur Formalisierung und Automatisierung von Change Plänen gesehen werden. In der Arbeit wird beschrieben, wie mithilfe von vollständig dokumentierten Abhängigkeiten zwischen IT-Services automatisiert ausführbare Workflows erstellt werden können. Es wird jedoch nicht beschrieben, wie Wartungsfehler während der Durchführung erkannt und vermieden werden können.

"ChangeLedge" ist ein weiteres Tool, das zur automatischen Erstellung von ausführbaren Change Plänen entwickelt wurde. Es unterscheidet sich von "CHAMPS", in dem es das Wissen aus bereits ausgeführten Wartungen nutzt, die in Form von sogenannten „Change Templates“ in einem Repository gesichert werden. Lücken in der Abhängigkeitsbeschreibung von IT-Services können durch den Einsatz dieser Change Templates umgangen werden [CMAS09].

Die bisher vorgestellten Arbeiten benötigen ein umfassendes Modell der bestehenden IT-Infrastruktur mit einer genauen Beschreibung der Abhängigkeiten zwischen den IT-Service-Komponenten, um automatisch Change Pläne erstellen zu können. Unter der Annahme, dass eine vollständige Beschreibung der Abhängigkeiten zwischen IT-Service-Komponenten nicht immer vorliegt und IT-Mitarbeiter skeptisch auf komplett automatisch erstellte Change Pläne reagieren, haben Trastour, Fink und Liu [TrFL09] "ChangeRefinery" entwickelt. "ChangeRefinery" bietet eine Unterstützung bei der Erstellung von Change Plänen, indem diese Schritt für Schritt von einem groben Change Plan in einen detaillierten Change Plan verfeinert werden können. Um bisher gesammelte Erfahrungen wiederzuverwenden, werden bereits durchgeführte IT-Service-Wartungen als Best Practices in einer Wissensdatenbank abgelegt und können bei der Verfeinerung neuer Change Pläne genutzt werden.

Auf Basis des Systems "CHAMPS" erläutert Keller, wie durch Formalisierung von Dokumenten, die im Rahmen des Change Management Prozesses verwendet werden, die automatisierte Wartungsplanung weiter verbessert werden kann. Dazu werden elektronische Verträge eingesetzt, die die unterschiedlichen Interessen der am Change Management Prozess beteiligten Stakeholder berücksichtigen. Keller schlägt hierfür die Formalisierung des RFCs[1] (Request for Change), der Deployment Deskriptoren, der Policys und Best Practices, sowie der Service Level Agreements vor. Deployment Deskriptoren beschreiben den Inhalt einer installierbaren Softwareeinheit, sowie der notwendigen Vorbedingungen, Abhängigkeiten und ausführenden Aktionen. Policys repräsentieren fachliche Ziele des Service-Anbieters,

[1] Ein RFC ist ein Antrag zur Durchführung einer Wartung. Weitere Details werden in Kapitel 2.6 beschrieben.

wie beispielsweise die Minimierung der Downtime. Unter Best Practices fasst Keller zusätzliche administrative Bedingungen zusammen, die bei der Durchführung der Wartung berücksichtigt werden müssen, zum Beispiel, dass bei einem Upgrade eines Application Servers in einem Cluster alle Application Server desselben Clusters ebenfalls upgegradet werden müssen [Kell05].

Um aus den Erfahrungen aus bereits durchgeführten IT-Service-Wartungen zu lernen, schlagen Cordeiro et al. [CMAW09] ein „Change Template Mining“-Verfahren vor, mit dem Log Files durchgeführter IT-Service-Wartungen mithilfe von Process Mining[2] Methoden analysiert werden und die Ergebnisse in Change Templates zusammengefasst werden. Diese Change Templates können mit dem entwickelten Tool "ChangeMiner" hinsichtlich Relevanz durchzuführender IT-Service-Wartungen untersucht und für die Erstellung von Change Plänen wiederverwendet werden.

Wickboldt et al. [WMCL09] haben den Prototyp "ChangeLedge" um eine Risikokomponente erweitert. In ihren Arbeiten stellen sie ein Konzept vor, mit dem Fehler bei der Durchführung von IT-Service-Wartungen analysiert werden und in Form von Log Records gespeichert werden können. Dabei unterscheiden sie zwischen verschiedenen Fehlerkategorien und analysieren die durch "ChangeLedge" erstellten Wartungspläne auf diese Fehlerkategorien, um daraus eine Risikokennzahl für jede Aktivität zu ermitteln [WBLA09]. Unter der Annahme, dass das Risiko mit jeder erneuten Durchführung einer IT-Service-Wartung aufgrund der Erfahrungswerte sinkt, ist das Risiko von neuen IT-Service-Wartungen dafür sehr hoch, da noch keine Erfahrungswerte zur Ausführung dieser expliziten Wartung vorliegen. Um auch gänzlich neue IT-Service-Wartungen auf ihr Risiko untersuchen zu können, haben Bianchin et al. eine Ähnlichkeitsmetrik zur Feststellung von Ähnlichkeiten zwischen Change Plänen entwickelt. Diese Metrik ermöglicht es, neue IT-Service-Wartungen bzw. Change Pläne auf Ähnlichkeit zu bereits durchgeführten IT-Service-Wartungen zu untersuchen, indem die durchzuführenden Aktivitäten verglichen werden und daraus das Risiko für Durchführungsfehler abgeleitet wird. Auf Basis der abgeleiteten Risiken können Maßnahmen zur Vorbeugung der Risiken getroffen werden, um nicht dieselben Fehler, die in der Vergangenheit gemacht wurden, zu wiederholen [BWGG10]. Die Ergebnisse der einzelnen Arbeiten sind in ein Framework zur automatisierten Risikoeinschätzung eingeflossen [WBLG11].

[2] Process Mining beschreibt Techniken zur Extraktion von Wissen aus Event Logs, um daraus Prozesse zu entdecken, zu überwachen und zu verbessern [AAMA11, S.170].

Automatisierte Konfigurationsprüfung

Um Konfigurationsfehler bei IT-Service-Wartungen zu vermeiden, wurden verschiedene Arbeiten vorgestellt, mit denen Konfigurationen vor der Durchführung einer IT-Service-Wartung auf ihre Korrektheit hin untersucht werden können. Dabei können die Ansätze in Regel-basierte Methoden und Online-Konfiguration-Validierungen unterschieden werden [XuZh15]. Arbeiten im Bereich der Systementwicklung mit dem Ziel, Systeme zu entwickeln, die keine Konfiguration erfordern, einfach zu konfigurieren sind oder robust gegen Konfigurationsfehler sind, werden nicht beschrieben.

Regel-basierte Methoden überprüfen Konfigurationsparameter gegen eine Menge vordefinierter Regeln, um Syntax- und Formatfehler aufzudecken [NOBM04]. Um Regeln systemübergreifend einzusetzen und auch logische Fehler aufdecken zu können, haben Huang et al. [HBSZ15] eine eigene Sprache zur Definition von Regeln zur Überprüfung von Konfigurationsparametern entwickelt. Zudem kann mit dieser Sprache der Aufwand zur Erstellung von Code zur Überprüfung von Konfigurationsparametern um den Faktor 10 reduziert werden (gemessen in Codezeilen).

Da die manuelle Definition von Regeln zur Überprüfung von Konfigurationen sehr aufwändig sein kann, schlagen einige Autoren lernende Systeme vor, die Konfigurations-Pattern von korrekt arbeitenden Systemen lernen und auf Basis dieser Pattern Regeln erstellen [XuZh15]. Dazu werden verschiedene Verfahren des maschinellen Lernens [YXPY11], extra dafür entwickelte Algorithmen [PLSW08] oder Clustering Algorithmen verwendet [KiWa04].

Jedoch lassen sich mit Regel-basierten Ansätzen semantische Konfigurationsfehler nur schwer ermitteln. Potenzial zur Aufdeckung semantischer Fehler bietet der Ansatz einer Online Validierung. Dabei wird eine Konfiguration in einer speziell dafür erstellten und von der Produktion isolierten Umgebung online getestet, um den Effekt bestimmter Konfigurationseinstellungen prüfen zu können [XuZh15]. Um aussagekräftige Testergebnisse zu erhalten, sollte die Validierungsumgebung dieselben oder zumindest sehr ähnliche Eigenschaften wie die tatsächliche Produktionsumgebung haben [BMNN05]. Damit eine ähnliche Auslastung wie im tatsächlichen Produktionssystem simuliert werden kann, schlagen Nagaraja et al. [NOBM04] und Oliveira et al. [ONBB06] vor, die Validierungsumgebung über einen Proxy-Server mit dem Produktionssystem zu verbinden und technische Nutzeranfragen mittels des Proxy-Servers zu duplizieren.

1.2.2 Während der Wartungsdurchführung

Die zuvor vorgestellten Arbeiten haben alle gemeinsam, dass versucht wird, potenzielle Durchführungsfehler durch eine entsprechende Planung und das Testen der Wartungsdurchführung a priori zu vermeiden. Jedoch können mit diesen Ansätzen Durchführungsfehler nicht gänzlich vermieden werden, da beispielsweise Fehler aufgrund von Unachtsamkeit der durchführenden Personen trotz sorgfältiger Planung auftreten können. Nachfolgend werden Forschungsarbeiten präsentiert, mit denen durch eine Überwachung der tatsächlichen Wartungsdurchführung Fehler entdeckt werden sollen, bzw. menschliche Fehler durch eine Automatisierung der Wartungsdurchführung verhindert werden sollen.

Monitoring von IT-Service-Wartungen

Xu et al. [XZWB14] haben die Methode "POD-Diagnosis" entwickelt, mit der es möglich ist, sporadische IT-Service-Wartungen in Cloud-Umgebungen, wie beispielsweise Upgrades, zu überwachen. Dazu werden die Logs der zu überwachenden IT-Service-Komponenten analysiert. Da Logs als einzige Informationsquelle zu wenig Informationsgehalt enthalten, werden zur Analyse Prozessmodelle herangezogen. Zunächst werden Prozessmodelle manuell oder per Process Mining erstellt. Anschließend werden die Logs mit Informationen aus dem Prozessmodell angereichert, zum Beispiel mit einer Prozess-, Instanzen- oder Prozessschritt-ID, und in einem separaten Log-Speicher abgelegt. Diese angereicherten Logs werden auf Konformität mit dem Prozessmodell untersucht und bei Abweichungen als Fehler markiert. Die Ergebnisse eines bestimmten Prozessschritts, z.B. die Verfügbarkeit einer Ressource oder eine bestimmte Konfiguration, werden mittels manuell angelegten Ergebnisausdrücken evaluiert. Für jeden Ausdruck wird ein Fehlerbaum für die Root-Cause-Analyse hinterlegt. Wenn eine Log Zeile als Fehler markiert wird, wird der Fehlerbaum des entsprechenden Ausdrucks analysiert, um die Ursache des Fehlers zu finden.

Dieser Ansatz wurde von den Autoren in Experimenten evaluiert, indem eine exemplarische IT-Service-Wartung durchgeführt wurde, und während der Durchführung zufällig acht verschiedene Fehler vorgenommen wurden. Das Ergebnis war mit einer Fehlererkennungsquote von über 90% vielversprechend, jedoch ist ein hoher manueller Aufwand erforderlich, um die Evaluationsausdrücke zur Überprüfung der Ergebnisse der einzelnen Prozessschritte zu spezifizieren und auszuwerten [FSWG15].

Farshchi et al. [FSWG17] haben eine Methode entwickelt, mit der Anomalien in der Wartungsdurchführung auf Basis von Regressionsmodellen identifiziert werden können und somit der manuelle Aufwand zur Spezifizierung von Ergebnisausdrücken deutlich geringer ausfällt. Als Datenquellen für die Regressionsmodelle werden Eventlogs und Metriken der

eingesetzten IT-Service-Komponenten verwendet, um daraus Korrelationen und potenzielle Kausalitäten abzuleiten. Die Eventlogs repräsentieren die Wartungsaktivitäten, wohingegen die Metriken den Zustand der IT-Service-Komponenten repräsentieren. Die Regressionsmodelle lernen aus korrekt durchgeführten IT-Service-Wartungen die Beziehung zwischen Eventlogs und Metriken und dienen als Vergleichsbasis, um Anomalien bei der Durchführung von IT-Service-Wartungen zu identifizieren.

Beiden Ansätzen ist gleich, dass die Aufdeckung von Anomalien zwar während der Wartungsdurchführung erfolgt, der Grund der Anomalie jedoch ex-post manuell analysiert werden muss. Die in dieser Arbeit vorgestellte Methode ähnelt der Methode „POD-Diagnosis“, nutzt jedoch als zusätzliche Datenquelle zu den Eventlogs nicht nur ein Prozessmodell, welches den Ablauf der IT-Service-Wartung beschreibt, sondern auch weitere Perspektiven, wie beispielsweise Zeit- und IT-Service-Komponenten-Aspekte. Zudem kann die Ursache einer Anomalie durch den Einsatz von Anti-Pattern direkt mit der Entdeckung eines Fehlers, ohne zusätzlichen Aufwand, bestimmt werden, wie es bereits auch in Compliance Anwendungen eingesetzt wird [LMMR13].

Ein anderer Ansatz zur Überwachung von IT-Service-Wartungen wird in [MGDA10] beschrieben. In dieser Arbeit wurde ein System entwickelt, das eine Überprüfung von Konfigurationsfehlern in einem Change Plan erlaubt, um Formatfehler bei Konfigurationen vor der Durchführung einer IT-Service-Wartung zu erkennen. Zusätzlich wurde ein System entwickelt, mit dem Kommandos an eine IT-Service-Komponente abgefangen und auf ihre Korrektheit überprüft werden können, z.B. ob der richtige Server heruntergefahren wird. Dieses System bietet den Vorteil, dass Aktivitäten in einer Wartungsdurchführung auf ihre Korrektheit mit dem Change Plan überprüft werden, bevor sie tatsächlich ausgeführt werden. Allerdings lassen sich damit keine Fehler erkennen, wenn das System bewusst oder unbewusst umgangen wird. Zudem werden keine Reihenfolgen von Aktivitäten überprüft, z.B. dass zuerst eine Aktivität a und anschließend erst Aktivität b durchgeführt werden darf.

Automatisierung der Wartungsdurchführung

Ein Ansatz zur Vermeidung von Durchführungsfehlern ist die Automatisierung der Durchführung von IT-Service-Wartungen. Der Trend „DevOps“ hat unter anderem die Automatisierung von bisher manuell durchgeführten Betriebsprozessen zum Ziel, um mit der Geschwindigkeit der agilen Softwareentwicklung mitzuhalten und manuelle fehleranfällige Tätigkeiten zu vermeiden [LwKO15]. Insbesondere in Unternehmen, die Internet-Anwendungen entwickeln, wie beispielsweise Facebook, werden vermehrt DevOps Methoden eingesetzt [FeFB13]. Im Rahmen der DevOps Bewegung sind bereits mehrere Open-Source

Tools zur Automatisierung von Softwarekonfigurationen und Softwareinstallationen entstanden, wie beispielsweise Puppet[3] oder Chef[4]. DevOps-Tools eignen sich insbesondere zur Automatisierung von End-to-End Installationen in Cloud Computing Umgebungen, benutzen jedoch jeweils eigene Artefakte und Runtime-Umgebungen, wodurch die Orchestrierung solcher Tools wiederum die Erstellung von Skripten und Workflows zur Übergabe von Parametern oder individueller Programmaufrufe erfordert [WeBL14]. Um die Orchestrierung dieser Tools zu vereinfachen, haben Wettinger et al. ein Framework auf Basis von TOSCA[5] (Topology and Orchestration Specification for Cloud Applications) entwickelt, mit dem verschiedene DevOps-Artefakte in TOSCA Modellen integriert werden können und somit IT-Services modelliert und automatisiert installiert werden können [WeBL14].

Die Automatisierung manueller Tätigkeiten kann die Anzahl menschlicher operativer Durchführungsfehler bei einer IT-Service-Wartung verringern. Vollständig ausgeschlossen werden können Wartungsfehler jedoch trotzdem nicht, da sich der menschliche Einfluss durch die Automatisierung von der eigentlichen Wartungsdurchführung auf die Wartungsplanung und Erstellung entsprechender Modelle bzw. Skripte verlagert. Zudem können auch bei einer automatisierten Wartungsdurchführung Fehler aufgrund externer Einflüsse, beispielsweise durch Hardware-Defekte oder zwischenzeitlich geänderter Software-Versionen, auftreten. Ein weiterer Nachteil solcher Tools ist, dass diese gut für lokale IT-Umgebungen funktionieren, die Erkennung von globalen Fehlern bei systemübergreifenden IT-Service-Wartungen jedoch aufgrund unterschiedlicher lokaler Fehlermeldungen nicht zufriedenstellend ist [XZWB14].

1.2.3 Beitrag zum aktuellen Forschungsstand

Diese Arbeit soll folgende Beiträge zum aktuellen Forschungsstand liefern:

- Im Vergleich zu bestehenden Arbeiten zur Überwachung von Wartungsdurchführungen und der Entdeckung von Anomalien, bei denen zur Identifikation des Fehlers ein Fehlerbaum erstellt werden muss, soll mit der in dieser Arbeit entwickelten Methode der Fehler direkt bei der Entdeckung einer Anomalie automatisiert ermittelt werden. Dadurch ist keine zusätzliche Erstellung eines Fehlerbaums mehr notwendig. Es sollen sowohl Ablauffehler als auch Konfigurationsfehler ermittelt werden.

[3] https://puppetlabs.com
[4] https://www.chef.io/chef/
[5] TOSCA ist ein von OASIS entwickelter Standard zur Beschreibung von Cloud-Anwendungen [Oasi13a].

- Mit der in dieser Arbeit entwickelten Methode sollen Wartungsfehler umgehend während der Wartungsdurchführung automatisiert ermittelt werden können, um diese beheben zu können, bevor sie sich in einem IT-Service manifestieren können.
- Für die automatisierte Wartungsdurchführung soll diese Arbeit einen Beitrag liefern, indem eine implementierungsunabhängige Modellierungssprache entwickelt wird. Mit dieser Sprache sollen alle Wartungsaktivitäten eines IT-Services modelliert werden können, um somit nicht nur lokale, sondern auch globale Fehler erkennen zu können. Zudem kann diese Sprache durch ihre Formalisierung potenziell zur Automatisierung von Wartungsaktivitäten eingesetzt werden.
- Da sich durch die Automatisierung von IT-Service-Wartungen der Einfluss menschlicher Fehler von der Wartungsdurchführung auf die Planung von IT-Service-Wartungen und dem Design von Change Plänen verschiebt, wird die Analyse von Change Plänen auf ihre Korrektheit an Bedeutung gewinnen. Für die Analyse von Change Plänen, um Wartungsfehler a priori zu vermeiden, soll in dieser Arbeit ein Beitrag geleistet werden, indem eine formalisierte Sprache zur Beschreibung von IT-Service-Wartungen vorgestellt wird.

1.3 Aufbau der Arbeit

Diese Arbeit ist wie folgt gegliedert:

In *Kapitel 2* wird eine Einführung in die Grundlagen des IT-Service-Managements gegeben. Zunächst werden dazu die Begriffe Service und IT-Service definiert, um anschließend Methoden des IT-Service-Managements zu beschreiben. Zum Abschluss des Kapitels wird der Prozess des Change Managements im Detail beschrieben, der die Planung, Steuerung und Kontrolle von IT-Service-Wartungen zum Gegenstand hat.

In *Kapitel 3* wird der Begriff des Fehlers definiert und von ähnlichen Begriffen wie IT-Service-Störung abgegrenzt. Anschließend werden Fehler, die bei der Durchführung von IT-Service-Wartungen in der betrieblichen Praxis auftreten, vorgestellt und kategorisiert. Zum Abschluss des Kapitel wird das allgemeine Prinzip der Fehlererkennung beschrieben, um darauf aufbauend das in dieser Arbeit verwendete Prinzip zur Fehlererkennung zu dokumentieren.

Anschließend werden in *Kapitel 4* die Grundlagen der Prozessmodellierung und der Modellierung von IT-Service-Komponenten beschrieben. Die Modellierung von Prozessen kann gemäß Oberweis [Ober96, S.20] als Grundlage zur Überwachung und Steuerung von

Prozessen dienen, um einen Soll-/ Ist-Vergleich durchzuführen und bei Abweichungen entsprechend reagieren zu können. In dieser Arbeit wird eine Kombination von XML-Netzen und TOSCA zur Modellierung von IT-Service-Wartungen verwendet. Diese Modelle werden zur Überwachung von IT-Service-Wartungen eingesetzt.

In *Kapitel 5* werden die Begriffe Pattern und Anti-Pattern eingeführt und eine Übersicht über verschiedene Arten von Pattern im Kontext betrieblicher Prozesse gegeben. Anschließend werden konkrete Pattern und Anti-Pattern vorgestellt, mit denen die in Kapitel 3 vorgestellten Kategorien von Fehlern bei der Durchführung von IT-Service-Wartungen entdeckt werden können. Pattern werden in dieser Arbeit eingesetzt, um zu beschreiben, wie eine IT-Service-Wartung ausgeführt werden soll. Anti-Pattern werden hingegen zur Laufzeit einer IT-Service-Wartung eingesetzt, um zu überprüfen, ob die tatsächliche Durchführung von der geplanten Durchführung der IT-Service-Wartung abweicht.

In *Kapitel 6* wird das Vorgehen zum Einsatz der Methode im Rahmen des Change Managements dokumentiert. Zudem wird beschrieben, wie Pattern aus dem Modell der IT-Service-Wartung instanziiert werden können. Hierfür werden zuvor die Grundlagen des Process Minings beschrieben, da zur Instanziierung der Pattern Methoden des Process Minings verwendet werden.

In *Kapitel 7* werden die Anforderungen an eine Architektur zur Umsetzung der Methode dokumentiert und die technische Umsetzung eines Software-Prototyps vorgestellt. Da für den Prototypen eine Complex Event Processing Engine verwendet wird, werden in diesem Kapitel Sprachkonzepte zur Verarbeitung von Ereignissen vorgestellt und die in Kapitel 5 eingeführten Anti-Pattern mittels dieser Sprachkonzepte ausgedrückt.

In *Kapitel 8* wird die Methode evaluiert. Dazu wird der Prototyp in drei verschiedenen Experimenten zur Überwachung von IT-Service-Wartungen eingesetzt. Während der Durchführung der IT-Service-Wartungen werden typische Fehler vorgenommen und gemessen, ob der Prototyp diese Fehler korrekt erkennt, und wieviel Zeit für die Ermittlung eines Fehlers benötigt wird.

Kapitel 9 schließt die Arbeit mit einer Zusammenfassung und Diskussion der Ergebnisse sowie einem Ausblick auf offene Forschungsfragen und weitere mögliche Entwicklungen.

Teile dieser Arbeit wurden bereits auf verschiedenen Tagungen vorgestellt und in Tagungsbänden, sowie Journals publiziert. Das Forschungsvorhaben selbst wurde in [Vett15] erstmalig vorgestellt. Die Methode zur Modellierung von IT-Service-Wartungen in Kapitel 4 wurde in [Vett16] veröffentlicht. Die Übersicht bekannter Pattern in Kapitel 5 basiert auf

[FKLS17] und [FKLS18]. Die in Kapitel 5 beschriebenen Pattern und Anti-Pattern, die in dieser Arbeit zum Einsatz kommen, sowie Ergebnisse aus Kapitel 7 und 8 wurden in [Vett17] und [Vett19] bereits veröffentlicht.

2 Grundlagen des IT-Service-Managements

In diesem Kapitel werden die Grundlagen des IT-Service-Managements beschrieben, insbesondere des Change Managements, welches das Einsatzgebiet und den Rahmen für die in dieser Arbeit vorgestellten Methode bildet. Hierfür wird zunächst allgemein der Begriff Service und im speziellen der Begriff IT-Service eingeführt und definiert. Anschließend werden die an einem IT-Service beteiligten Rollen beschrieben und definiert. Darauf aufbauend wird der Begriff IT-Service-Management definiert. Am Beispiel des De-facto-Standards „ITIL“, werden die eingesetzten Methoden im IT-Service-Management beschrieben. Eine der zentralen Methoden im IT-Service-Management bildet das Change Management, welches abschließend detailliert beschrieben wird.

2.1 Service

Trotz der starken Verbreitung und Nutzung des Begriffs „Service“ in verschiedenen Forschungsgebieten hat sich bisher keine eindeutige Definition des Begriffs etabliert [Göss05]. So sind in den verschiedenen Forschungsgebieten unterschiedliche Interpretationen des Begriffs „Service“ vorzufinden [BaGO04]. In der Informatik werden mit dem Begriff „Service“ vor allem technologische Aspekte betrachtet, zum Beispiel Web Services oder eine Serviceorientierte Architektur. Aus der betriebswirtschaftlichen Perspektive wird ein Service als eine Dienstleistung verstanden [BHHK08]. In der betriebswirtschaftlichen Forschung, können Definitionen von Services in folgende drei Kategorien eingeordnet werden [MeBr06]:

- Enumerative Definitionen
  Bei enumerativen Definitionen wird versucht, einen Service durch die Aufzählung von Beispielen zu präzisieren, beispielsweise durch die Aufzählung von verschiedenen Wirtschaftsbereichen wie Beherbergung, Energieversorgung, Erholung, Forschung, Information, Nachrichtenübermittlung, Rechts- & Wirtschaftsberatung, Unterhaltung, etc. [Klei98]. Eine solche Definition kann als Präzisierung eines Begriffs verwendet werden, ist aus einer wissenschaftlichen Betrachtungsweise jedoch noch zu grob, da beispielsweise manche Wirtschaftsbereiche nicht eindeutig

oder nur teilweise dem Service-Sektor zugeordnet werden können und daher keine präzise Trennung zu anderen Wirtschaftssektoren möglich ist [Klei98].

- Negativdefinitionen
  Bei Negativdefinitionen wird ein Service dadurch definiert, in dem aufgezählt wird, was ein Service nicht ist. Dazu werden häufig Sachleistungen genannt, um diese von Services zu differenzieren [BuSt06]. Für eine präzise wissenschaftliche Definition ist eine Negativdefinition jedoch ebenfalls nur bedingt geeignet, da keine Merkmale eines Services herausgearbeitet werden, die eine Abgrenzung zu Sachgütern, oder auch Mischformen von Services und Sachleistungen ermöglichen [BuSt06].
- Konstitutive Definitionen
  Bei konstitutiven Definitionen wird ein Service durch die Aufzählung von Merkmalen eines Services definiert [BaGO04]. Konstitutive Definitionen lassen sich wiederum unterteilen in potenzialorientierte, prozessorientierte und ergebnisorientierte Ansätze [BuSc06]:
  Potenzialorientierte Ansätze heben hervor, dass der Anbieter eines Services das Potenzial haben muss, die Anforderungen des Abnehmers zu befriedigen. Er muss somit die für den Service notwendigen technischen und/oder menschlichen Ressourcen bereithalten. Der Service-Anbieter gibt somit ein Leistungsversprechen gegenüber dem Abnehmer ab. Potenzialorientierte Ansätze betonen das konstitutive Merkmal der „Immaterialität“ eines Services [BuSc06].
  Prozessorientierte Ansätze definieren Services als Prozess zwischen dem Service-Anbieter und dem Service-Abnehmer. Die Produktion und der Konsum des Services finden simultan statt [ChSp06]. Prozessorientierte Ansätze heben daher als konstitutives Merkmal das Uno-actu-Prinzip hervor. Das bedeutet, dass nach dieser Definition nur das Versprechen eines Services und die Bereithaltung der dafür notwendigen Ressourcen noch kein Service darstellt. Der Service entsteht erst durch die Einlösung des Versprechens und die tatsächliche Nutzung durch den Service-Abnehmer.
  Ergebnisorientierte Definitionsansätze betonen nicht den Prozess der Service-Erstellung, sondern die Wirkung des Ergebnisses auf den Service-Abnehmer [Male13]. Konstitutive Merkmale, die bei ergebnisorientierten Definitionsansätzen hervorgehoben werden, sind somit die Immaterialität, sowie die Einbeziehung des Service-Abnehmers (externer Faktor) in die Service-Erstellung [BHHK08].

Für eine Übersicht zu Definitionen, die den verschiedenen Definitionsansätzen zugeordnet werden können, sei an dieser Stelle auf Leimeister [Leim12] verwiesen. Abbildung 2.1 veranschaulicht die verschiedenen konstitutiven Definitionsansätze und die daraus abgeleiteten Merkmale von Services.

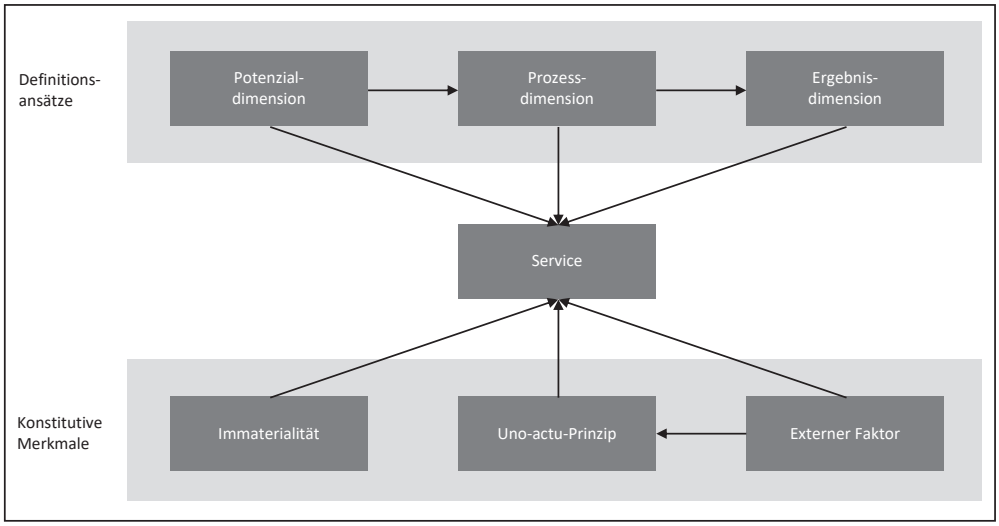


Abbildung 2.1: Dimensionen und konstitutive Merkmale eines Service

Die verschiedenen Definitionsansätze sind nicht konträr zueinander zu sehen, sondern komplementär zueinander. Beispielsweise kann ein Service allein durch seine Potenzialdimension definiert werden. Wird dieser durch die Prozessdimension definiert, wird die Potenzialdimension automatisch inkludiert, da für die Durchführung eines Prozesses die dafür notwendigen Human- und/ oder technischen Ressourcen vorgehalten werden müssen. Die Ergebnisdimension hingegen erweitert die Prozessdimension um die Betrachtung des Prozessergebnisses auf den Service-Abnehmer. Die verschiedenen Dimensionen können daher als Abschnitte eines Service-Lebenszyklus interpretiert werden. Für die Definition eines Service im Rahmen dieser Arbeit sollen daher alle Dimensionen und Merkmale Berücksichtigung finden:

Definition 2.1: Service, Dienstleistung

Ein Service ist zunächst ein Versprechen eines Service-Anbieters an einen potenziellen Service-Abnehmer, welches später in einem betrieblichen Prozess ausgeführt

und gleichzeitig verbraucht wird, um dabei sofort oder verzögert eine Wirkung auf den Service-Abnehmer zu entfalten.

■

Da in dieser Arbeit Services im betrieblichen Umfeld behandelt werden, wird der Prozess der Service-Erbringung als betrieblicher Prozess verstanden. Der Definition von Oberweis [Ober96, S.14f.] folgend wird ein betrieblicher Prozess bzw. betrieblicher Ablauf[1] wie folgt definiert:

Definition 2.2: Betrieblicher Prozess

Ein betrieblicher Prozess „ist eine Menge von manuellen, teil-automatisierten oder automatisierten Aktivitäten, die in einem Betrieb nach bestimmten Regeln auf ein bestimmtes Ziel hin ausgeführt werden“ [Ober96, S.14f.].

■

Aktivitäten werden durch Aufgabenträger ausgeführt. Diese können unterschieden werden in personelle und maschinelle Aufgabenträger [FeSi08, S.3, Ober96, S.15]. Als maschinelle Aufgabenträger werden Rechner- und Kommunikationssysteme verstanden, die um System- und Anwendungssoftware erweitert sind, um die auszuführenden Aktivitäten übernehmen zu können. Die um Software erweiterte Hardware wird Anwendungssystem genannt, und bildet den eigentlichen maschinellen Aufgabenträger [FeSi08, S.4f.].

Nachdem der Begriff des allgemeinen Service definiert ist, wird nachfolgend eine Konkretisierung der Ausprägung „IT-Service“ vorgenommen.

2.2 IT-Service

Um den Begriff „IT-Service“ vom allgemeinen „Service“-Begriff zu unterscheiden, typisiert Leimeister [Leim12] Services anhand ihrer Umsetzung in:

- rein personenbasierte Services,
- mechanische Services,
- IT-unterstütze Services oder
- IT-Services.

[1] Die Begriffe betrieblicher Prozess und betrieblicher Ablauf werden in dieser Arbeit synonym verwendet.

Rein personenbasierte Services sind Services, die durchgehend von Menschen erbracht und konsumiert werden. Mechanische Services sind Dienstleistungen, die starken Gebrauch von mechanischen Geräten machen, z.B. Kaugummiautomaten. Bei IT-Services erfolgt die Service-Erbringung automatisiert durch maschinelle Aufgabenträger. IT-Services können beispielsweise angebotene Web-Services sein, die durch ein Anwendungsprogramm automatisiert aufgerufen werden. Bei IT-unterstützen Services erfolgt die Service-Erbringung teilautomatisiert. Ein Beispiel für einen IT-unterstützten Service ist das Identifikationsverfahren bei einer Bankkontoeröffnung per Video-Chat [Leim12].

Diese Typisierung wird in dieser Arbeit aufgegriffen, um den Begriff IT-Service einzugrenzen. Im vorigen Kapitel wird ein Service definiert als „[...] ein Versprechen [...], welches später in einem betrieblichen Prozess ausgeführt und gleichzeitig verbraucht wird [...]“. Wie von Leimeister [Leim12] beschrieben, können Services anhand ihrer Art der Erbringung klassifiziert werden. Diesen Vorschlag aufgreifend, wird ein IT-Service anhand der Menge maschineller Aufgabenträger, im Prozess der Service-Erbringung, von anderen Services abgegrenzt. Ein Service könnte insofern als rein personenbasierter, IT-unterstützter oder IT-Service realisiert werden. Bei IT-Services werden alle Aktivitäten durch das Zusammenspiel technischer Systemkomponenten, wie beispielsweise vernetzte Speicher- und Serversysteme ausgeführt [BöKr04]. Dabei ist zu beachten, dass ein Service auch aus anderen Services aufgebaut werden kann. Ein Beispiel hierfür sind Mashups [HoSt09], bei denen angebotene Services von Dritt-Service-Anbietern zu einem übergeordneten Service integriert werden [KoTP09]. Ein anderes Beispiel ist die Bereitstellung einer Software-as-a-Service-Anwendung eines Service-Anbieters, die wiederum auf Infrastructure- oder Platform-as-a-Service-Diensten eines anderen Anbieters aufbaut. Ein IT-Service kann beispielsweise auch mit einem personenbasierten Service zu einem IT-unterstützten Service integriert werden. Anders herum betrachtet kann dieser IT-unterstützte Service in einen IT-Service und einen personenbasierten Service modularisiert werden. Unter der Modularisierung eines Services wird die Zerlegung des Services in seine Bestandteile, sogenannte Module, verstanden [BöKr04]. Abbildung 2.2 veranschaulicht die Modularisierung eines IT-unterstützten Services in einen personenbasierten und einen IT-Service.

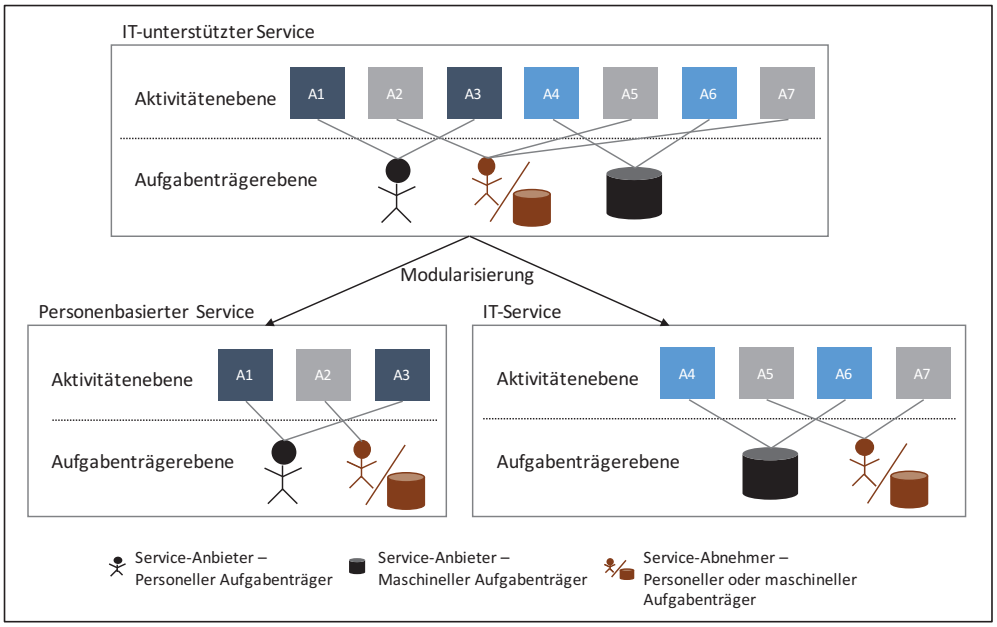


Abbildung 2.2: Beispielhafte Modularisierung eines IT-unterstützen Services

Aufgrund der Modulisierbarkeit von Services und somit der Dekomposition von IT-unterstützten Services, unter anderem in einzelne IT-Services, wird ein IT-Service wie folgt definiert:

Definition 2.3: IT-Service

Ein IT-Service ist ein Service, bei dem die Aktivitäten des Service-Anbieters bei der Service-Erbringung vollständig durch maschinelle Aufgabenträger ausgeführt werden. Die zur Service-Erbringung durchzuführenden Aktivitäten des Service-Abnehmers können sowohl von personellen als auch von maschinellen Aufgabenträgern ausgeführt werden.

■

Ein Beispiel für einen IT-Service ist Google Maps[2]. Der Service-Anbieter ist Google. Die Erbringung des IT-Services erfolgt auf Seiten von Google durch maschinelle Aufgabenträger. Ein IT-Service Abnehmer kann ein personeller Aufgabenträger sein, der beispielsweise manuell einen Ort eingibt, um sich zu diesem navigieren zu lassen. Der IT-Service-Abnehmer

[2] https://www.google.de/maps/

kann aber auch ein maschineller Aufgabenträger sein. Ein Beispiel hierfür ist, wenn eine Applikation die Koordinaten eines Ortes anzeigt und hierfür die API von Google Maps aufruft, um die diese zu ermitteln.

Die Anwendungen, die zur Ausführung der Aktivitäten des Service-Anbieters eingesetzt werden, werden in dieser Arbeit als IT-Service-Komponenten bezeichnet.

Definition 2.4.: IT-Service-Komponente

Eine IT-Service-Komponente ist die Anwendung (Software) eines maschinellen Aufgabenträgers, der zur Ausführung der Aktivitäten des Service-Anbieters eingesetzt wird.

■

2.3 IT-Service-Rollen

Allgemein lassen sich bei der Beteiligung an einem IT-Service die zwei Rollen IT-Service-Anbieter und IT-Service-Abnehmer unterscheiden [GHKR01]. Die Hauptaufgabe des IT-Service-Anbieters ist die Bereitstellung des IT-Services. Die Rolle des IT-Service-Abnehmers kann entsprechend seiner Hauptaufgaben nochmals unterschieden werden in die Rollen des IT-Service-Nutzers und des IT-Service-Kunden. Der IT-Service-Nutzer konsumiert den IT-Service. Der IT-Service-Kunde hat ein Interesse an der Nutzung des IT-Services und verantwortet daher alle Aktivitäten zur Verwaltung des Services, wie beispielsweise die Vertragsgestaltung [GHKR01]. Bartsch [Bart10] definiert einen IT-Service-Nutzer als einen personellen Aufgabenträger. Diese Sicht wird in dieser Arbeit nicht geteilt, da ein IT-Service „A“ auch durch einen IT-Service „B“ genutzt werden kann und erst IT-Service „B“ durch einen personellen Aufgabenträger genutzt wird (beispielsweise Mashups, siehe Kapitel 2.2). Abbildung 2.3 stellt die an einem IT-Service beteiligten Rollen graphisch dar.

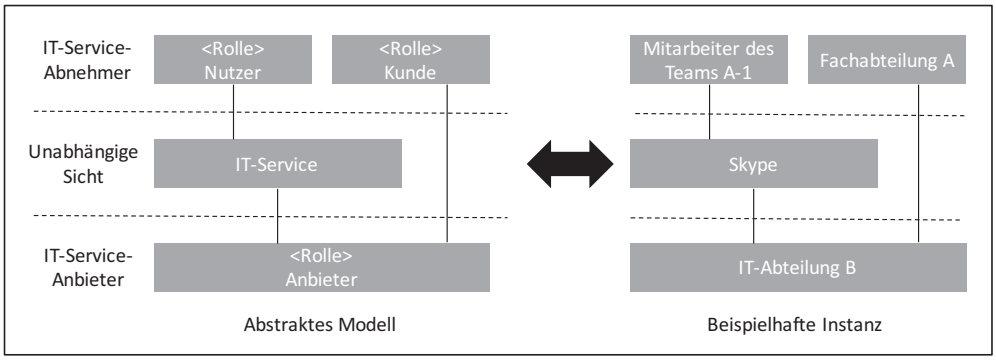


Abbildung 2.3: Beteiligte Rollen an einem IT-Service (angelehnt an [GHKR01])

Definition 2.5: IT-Service-Abnehmer

Ein IT-Service-Abnehmer nutzt einen IT-Service von einem IT-Service-Anbieter. Der IT-Service-Abnehmer kann eine natürliche Person, als auch eine Organisation sein, und kann wiederum in die Rollen eines IT-Service-Nutzers und eines IT-Service-Kunden unterschieden werden.

■

Definition 2.6: IT-Service-Nutzer

Der IT-Service-Nutzer kann ein personeller oder maschineller Aufgabenträger sein, der den eigentlichen IT-Service zur Erfüllung bestimmter Leistungen nutzt.

■

Definition 2.7: IT-Service-Kunde

Der IT-Service-Kunde ist ein personeller Aufgabenträger, der alle notwendigen Management-Aktivitäten auf Seiten des IT-Service-Abnehmers durchführt, damit der IT-Service-Nutzer den IT-Service nutzen kann.

■

IT-Service-Anbieter und IT-Service-Abnehmer können sowohl in einer anonymen Beziehung als auch in einer organisatorischen Beziehung zueinander stehen [Bren07]. Eine anonyme Beziehung besteht beispielsweise beim Bezug eines öffentlichen Cloud-Services. Die beiden

Geschäftspartner sind in diesem Fall nur durch ein Service-Level-Agreement (SLA)[3] miteinander verbunden. Eine organisatorische Beziehung besteht, wenn der Service-Anbieter und der Service-Abnehmer Teil einer gemeinsamen Organisation sind und die Beziehung somit über mehr als nur ein SLA definiert ist. Entsprechend wird von einem externen bzw. internen Service-Anbieter gesprochen [Bren07], [BoJK08].

Definition 2.8: IT-Service-Anbieter

Ein IT-Service-Anbieter führt alle notwendigen Management- und Leistungsaktivitäten zur Bereitstellung und Ausführung eines IT-Services durch, damit dieser von einem IT-Service-Abnehmer verwendet werden kann. Bietet ein IT-Service-Anbieter seine IT-Services nur einem IT-Service-Abnehmer innerhalb derselben Organisation an, wird dieser als interner IT-Service-Anbieter bezeichnet. Bietet der IT-Service-Anbieter seine IT-Services auch IT-Service-Abnehmern außerhalb seiner Organisation an, wird dieser als externer IT-Service-Anbieter bezeichnet.

■

Allgemein lässt sich der IT-Bereich eines IT-Service-Anbieters in die zwei Kernaufgabenbereiche Anwendungsentwicklung und Anwendungsbetrieb unterscheiden [Krcm15]. Je nach organisatorischer Größe können diese durch weitere Aufgabenbereiche erweitert werden [Krcm15]. Die Anwendungsentwicklung ist für die Entwicklung von Anwendungen bzw. die Auswahl und Anpassung entsprechender Standard-Software verantwortlich. Dafür erhält diese einen entsprechenden Auftrag von dem anfordernden Fachbereich. Der Anwendungsbetrieb ist für den Betrieb der von der Anwendungsentwicklung übergebenen Anwendung verantwortlich und stellt diese dem Auftraggeber zur Nutzung zur Verfügung. Die Bereitstellung der Anwendung kann dabei in Form eines SLAs festgehalten werden. Abbildung 2.4 stellt den klassischen Aufbau des IT-Bereichs eines internen Service-Anbieters beispielhaft dar.

[3] Ein SLA ist ein Vertrag bzw. eine Abmachung zwischen einem IT-Service-Anbieter und einem IT-Service-Abnehmer, in dem für einen festgelegten Preis ein vereinbarter Service unter vereinbarten Bedingungen, definiert wird [LeBe02].

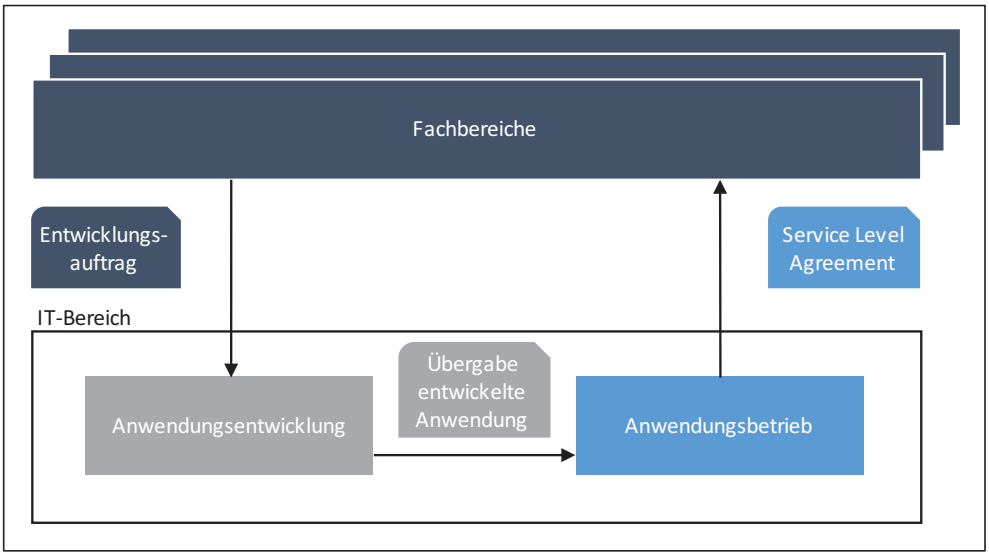


Abbildung 2.4: Strukturierung eines unternehmensinternen IT-Bereichs (angelehnt an [Bren07])

2.4 IT-Service-Management

Um die gestiegene Komplexität von IT-Services und deren Betrieb zu beherrschen, hat sich das Konzept des IT-Service-Managements (ITSM) entwickelt [WiCE09]. Das IT-Service-Management hat das Management der IT-Betriebsprozesse zum Forschungsgegenstand [GDQC09]. Eine wichtige Eigenschaft des ITSMs ist die Fokussierung auf Prozesse und die ständige Verbesserung dieser [GaDa10], um die Geschäftsziele und Bedürfnisse des IT-Service-Abnehmers zu erfüllen [CoWE08].

Definition 2.9: IT-Service-Management

IT-Service-Management umfasst Methoden und Prozesse, um die von einem IT-Service-Anbieter erbrachten IT-Services auf die Bedürfnisse und Geschäftsziele des IT-Service-Abnehmers auszurichten.

■

Das IT-Service Management zeichnet sich durch eine Markt-, Service-, Lebenszyklus- und Prozessorientierung aus [ZaHB06]:

- Marktorientierung
  Im IT-Service-Management nehmen der IT-Bereich eines Unternehmens und der Fachbereich nicht mehr die Rollen von Projektpartnern ein, sondern von Lieferant und Kunde. Die Lieferanten-Kunden-Beziehung zwischen dem IT-Bereich und dem Fachbereich wird durch marktorientierte Vertragsbeziehungen gesteuert.
- Serviceorientierung
  Die Geschäftsgrundlage zwischen IT-Lieferanten und ihren Kunden sind IT-Services. Die IT-Service-Anbieter führen ein Angebotsportfolio, aus dem sich die Kunden bedienen können, um ihre Bedürfnisse zu befriedigen. Das klassische Projektportfolio eines IT-Bereichs wandelt sich somit beim IT-Service-Management zu einem Leistungsportfolio.
- Lebenszyklusorientierung
  Im traditionellen IT-Management liegt der Fokus hauptsächlich auf der Entwicklung von Anwendungen. So werden beispielsweise bei der Kostenkalkulation primär die Entwicklungskosten betrachtet. Im IT-Service-Management wird nicht nur die Anwendungsentwicklung, sondern der gesamte Lebenszyklus eines IT-Services, von der Planung, der Entwicklung, dem Betrieb und der Wartung, bis zur Abschaltung eines IT-Services betrachtet.
- Prozessorientierung
  IT-Bereiche sind traditionell eher funktional als prozessorientiert ausgerichtet. Im IT-Service-Management erfolgt eine Ausrichtung des IT-Bereichs anhand der zur Erbringung eines IT-Services relevanten Prozesse, anstatt an den funktionalen Einheiten.

Es gibt verschiedene ITSM-Frameworks auf dem Markt. Das verbreitetste Framework und der De-facto-Standard in der Industrie ist ITIL [HoZB04]. Aufbauend auf ITIL wurden weitere ITSM-Frameworks wie beispielsweise der International Organization for Standards (ISO)/ International Electrotechnical Commission (IEC) 20000 Standard (ISO/IEC 20000) oder Hersteller-spezifische Frameworks wie das Process Reference Model for IT (PRM-IT) von IBM und Microsoft Operations Framework (MOF) von Microsoft entwickelt [GDQC09]. Aufgrund des Status als De-facto-Standard im ITSM, werden nachfolgend die grundlegenden Konzepte des ITSM am Beispiel von ITIL beschrieben.

2.5 IT Infrastructure Library (ITIL)

In ITIL wird das IT-Service-Management in Form eines IT-Service-Lebenszyklus, bestehend aus den fünf Phasen *Service Strategy*, *Service Design*, *Service Transition*, *Service Operation*

und *Continual Service Improvement*, strukturiert. Die ITIL Edition 2011 besteht aus fünf Hauptpublikationen, den sogenannten „ITIL Core"-Publikationen, sowie weiteren ergänzenden Publikationen. Die fünf Hauptpublikationen sind:

- ITIL Service Strategy [Offi11b],
- ITIL Service Design [Offi11c],
- ITIL Service Transition [Offi11a],
- ITIL Service Operation [Offi11d],
- ITIL Continual Service Improvement [Offi11e]

und repräsentieren im IT-Service-Lebenszyklus jeweils eine Phase. Innerhalb jeder Phase werden verschiedene Prozesse beschrieben. Diese Prozesse sind nicht isoliert zu betrachten, sondern stets im Kontext des Lebenszyklus mit Schnittstellen zu anderen Prozessen [BeZi14]. So finden manche Aktivitäten auch über verschiedene Phasen des Lebenszyklus statt [RaGr12] In Abbildung 2.5 wird der Lebenszyklus, inklusive der verschiedenen Prozesse in den jeweiligen Phasen dargestellt. Nachfolgend werden die einzelnen Phasen und die darin enthaltenen Prozesse beschrieben.

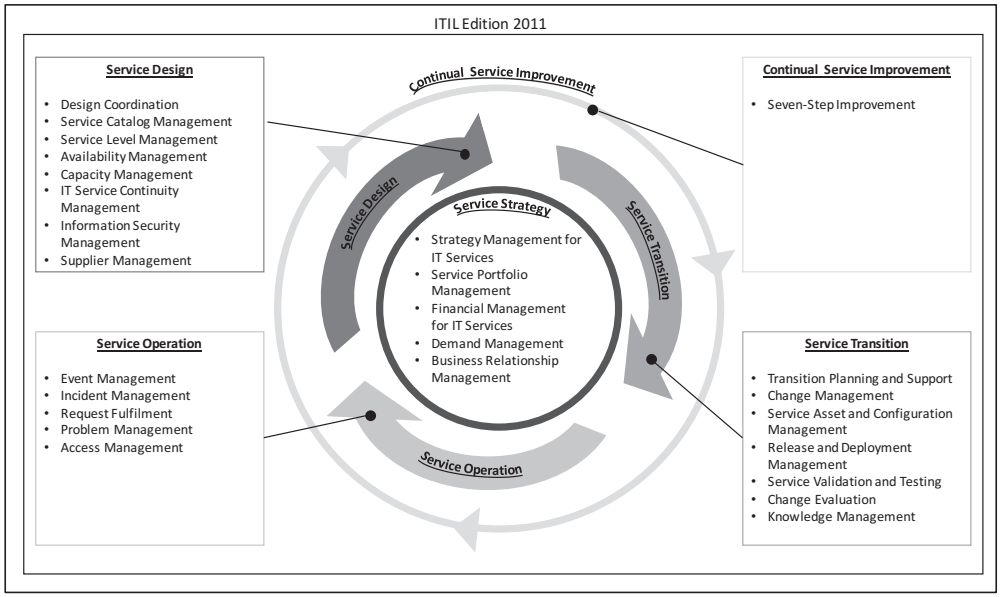


Abbildung 2.5: ITIL Service-Lebenszyklus inklusive Prozesse (angelehnt an [Offi11a])

2.5.1 Service Strategy

In der Publikation ITIL Service Strategy [Offi11b] wird die Phase *Service Strategy* beschrieben, die im Zentrum des IT-Service-Lebenszyklus eingeordnet ist. IT-Service-Management wird nicht als ein Bündel organisatorischer Maßnahmen aufgefasst, sondern als strategisches Werkzeug. Entsprechend werden in der Publikation Richtlinien, Prozesse und Anleitungen zum Design, der Entwicklung und der Implementierung eines IT-Service-Managements aus strategischer Sicht beschrieben. Ausgangspunkt ist die Analyse der organisatorischen Ziele und das Verstehen der Kundenbedürfnisse, um IT-Services anbieten zu können, mit denen diese Ziele erreicht und die Kundenbedürfnisse befriedigt werden können. Im Einzelnen werden folgende Prozesse beschrieben:

- Strategy Management for IT Services
  Gegenstand dieses Prozesses ist die generelle strategische Ausrichtung der IT-Service-Management-Aktivitäten.
- Service Portfolio Management
  Ziel des *Service Portfolio Managements* ist die Verwaltung eines Portfolios an Services, das, entsprechend den finanziellen Vorgaben, dem Kunden zur Befriedigung seiner Bedürfnisse bereitgestellt werden kann.
- Financial Management for IT Services
  Im Rahmen dieses Prozesses werden die finanziellen Ressourcen für das IT-Service-Management verwaltet. Dazu gehört sowohl die Budgetierung für genutzte Services, als auch die generelle Kosten- und Leistungsverrechnung.
- Demand Management
  Gegenstand des *Demand Managements* ist die Analyse des Kundenbedarfs an einem Service, so dass dieser in ausreichender Kapazität bereitgestellt werden kann.
- Business Relationship Management
  Das Ziel dieses Prozesses ist der Aufbau und Erhalt einer positiven Kundenbeziehung, sowie die Sicherstellung, dass die Kundenbedürfnisse des Kunden durch die angebotenen oder zukünftigen Services befriedigt werden.

2.5.2 Service Design

Die nächste Phase im IT-Service-Lebenszyklus ist die Phase *Service Design* [Offi11c], in der IT-Service-Portfolios festgelegt werden, um die strategischen Ziele, die in der Phase Service Strategy festgelegt werden, umzusetzen. Dies beinhaltet auch die Veränderung von beste-

henden IT-Services[4], um sie zu verbessern bzw. den geänderten Kundenbedürfnissen anzupassen. Im Service Design werden auch Richtlinien für den Entwurf von IT-Services definiert, sowie SLAs definiert. Im Einzelnen werden folgende Prozesse beschrieben:

- Design Coordination
  Dieser Prozess ist für die Koordination aller Ressourcen, Aktivitäten und Prozesse in der Phase *Service Design* verantwortlich.
- Service Catalog Management
  Dieser Prozess beschreibt die Erstellung und den Betrieb eines IT-Servicekatalogs. Ein IT-Servicekatalog kann ein strukturiertes Dokument oder eine Datenbank sein, in der alle IT-Services beschrieben sind und dem Kunden zur Information zur Verfügung gestellt werden.
- Service Level Management
  Im Prozess *Service Level Management* werden SLAs und angemessene Service Level Ziele verhandelt und auf ihre Einhaltung hin kontrolliert.
- Availability Management
  Im *Availability Management* wird verantwortet, dass alle notwendigen Ressourcen und Prozesse zur Sicherstellung der Verfügbarkeit von IT-Services, entsprechend den vereinbarten Service Level Zielen, bereitgestellt werden.
- Capacity Management
  Das *Capacity Management* erweitert das Availability Management, so dass zu jeder Zeit sichergestellt ist, dass der IT-Service nicht nur verfügbar, sondern auch von der Kapazität den Anforderungen des Kunden genügt.
- IT Service Continuity Management
  Dieser Prozess hat die Analyse von Risiken und die Definition von Maßnahmen zum Gegenstand, um den Betrieb von IT-Services auch im Schadensfall gewährleisten zu können.
- Information Security Management
  In diesem Prozess werden alle IT-Sicherheitsfragen bezüglich den IT-Services gehandhabt. Der Prozess *Information Security Management* ist in vielen Unternehmen Teil eines gesamtbetrieblichen Ansatzes und nicht beschränkt auf das IT-Service-Management.

[4] In der ITIL wird ein IT-Service als eine Kombination von Personen, Prozessen und IT, die von einem IT-Service-Provider für den Kunden bereitgestellt wird, definiert [Offi11a]. ITIL fasst den Begriff des IT-Services somit weiter, als er in dieser Arbeit definiert ist. Der Definition in dieser Arbeit folgend handelt es sich um einen IT-unterstützen Service. Aus Gründen der Lesbarkeit wird jedoch zur Beschreibung der ITIL der Begriff IT-Service, statt IT-unterstützter Service, verwendet.

- Supplier Management
  Gegenstand des *Supplier Managements* ist die Sicherstellung und Kontrolle, dass Verträge mit Lieferanten so gestaltet sind, dass IT-Services ohne Unterbrechungen betrieben werden können, und alle Lieferanten ihre vertraglich festgehaltenen Pflichten einhalten.

2.5.3 Service Transition

Die Phase *Service Transition* ist die Schnittstelle zwischen der Phase *Service Design* und dem Betrieb von IT-Services in der Phase *Service Operation*. In der entsprechenden Hauptpublikation [Offi11a] werden Prozesse beschrieben, um unter Berücksichtigung der strategischen Vorgaben und der Design-Richtlinien neue IT-Services oder Änderungen an bestehenden IT-Services geregelt in den Betrieb zu überführen. In der Phase *Service Transition* wird zudem der Prozess *Change Management* beschrieben. Die grundlegende Methode des Change Managements[5] wird aufgrund des Forschungsgegenstands dieser Arbeit in Kapitel 2.6 ausführlich beschrieben. Im Einzelnen werden folgende Prozesse in der Phase *Service Transition* beschrieben:

- Transition Planning and Support
  Dieser Prozess ist für die Planung aller Prozesse in der Phase *Service Transition* und der Koordination der dafür benötigten Ressourcen verantwortlich.
- Change Management
  Das *Change Management* hat die Planung, Steuerung und Kontrolle von Changes und somit von Wartungen zum Gegenstand und wird aufgrund des Forschungsgegenstands dieser Arbeit in Kapitel 2.6 im Detail beschrieben.
- Service Asset and Configuration Management
  Das Ziel des *Service Asset and Configuration Management* ist die Verwaltung und Dokumentation aller zur Erbringung eines IT-Services notwendigen Vermögenswerte und IT-Service-Komponenten, sowie ihrer Konfigurationen und Abhängigkeiten zueinander. Zur Verwaltung und Dokumentation dieser Dokumente schlägt ITIL die Nutzung eines Configuration Management Systems (CMDB) vor, das aus mehreren Configuration Management Databases (CMDB) bestehen kann.

[5] In der betriebswirtschaftlichen Forschung wird der Begriff *Change Management* für die Disziplin des organisatorischen Veränderungsmanagements verwendet (für weitere Informationen siehe [By05]). In ITIL und in dieser Arbeit wird mit dem Begriff *Change Management* das Management der Änderung von IT-Service verstanden.

- Release and Deployment Management
  Der Prozess *Release and Deployment Management* ist für die Planung, Steuerung und Kontrolle des Übergangs eines Releases in eine Umgebung verantwortlich.
- Service Validation and Testing
  Gegenstand des Prozesses *Service Validation and Testing* ist die Validierung und das Testen eines geänderten oder neuen IT-Services, um sicherzugehen, dass der IT-Service die Designspezifikationen erfüllt und den Kundenbedürfnissen gerecht wird.
- Change Evaluation
  Change Evaluation dient der Bewertung eines geänderten oder neuen IT-Services, um festzustellen, ob dieser im bewerteten Zustand in Betrieb genommen werden soll.
- Knowledge Management
  Der Prozess *Knowledge Management* dient der Sammlung und Analyse von Informationen, um bereits vorhandenes Wissen nicht neu zu entwickeln und somit die Effizienz bei der Bereitstellung und dem Betrieb von IT-Services zu erhöhen.

2.5.4 Service Operation

In der Phase *Service Operation* [Offi11d] werden Prozesse zum Betrieb von IT-Services beschrieben. Dabei ist sicherzustellen, dass die im Betrieb befindlichen IT-Services die mit dem IT-Service-Abnehmer vereinbarten Service Level einhalten. Hierzu zählt die Sicherstellung eines reibungslosen Ablaufs der IT-Services, sowie die Wiederherstellung und der Support von IT-Services, wenn diese ausfallen sollten. Es werden folgende Prozesse aufgeführt:

- Event Management
  Das *Event Management* dient der Überwachung von IT-Services, um auftretende Ereignisse, wie beispielsweise den ungeplanten Ausfall eines Servers, zu erkennen und entsprechende Maßnahmen zu ergreifen.
- Incident Management
  Im Prozess *Incident Management* wird der Ablauf zur Wiederherstellung eines ungeplant ausgefallenen IT-Services oder im Falle einer Qualitätsminderung eines IT-Services die Wiederherstellung der Qualität eines IT-Services koordiniert.
- Request Fulfillment
  Im Rahmen des Prozesses *Request Fulfillment* werden Anfragen von IT-Service-Nutzern, hinsichtlich Informationen oder einer Beratung zu einem IT-Service, einer Veränderung eines IT-Services, oder dem Zugriff auf einen IT-Service koordiniert.

- Problem Management
  Das *Problem Management* ist ein Prozess zur Behebung oder Verminderung von Problemen, die zu Störfällen bei IT-Services führen. Dabei werden Probleme im gesamten IT-Service-Lebenszyklus von der Strategie bis zum Betrieb untersucht.
- Access Management
  Das *Access Management* ist dafür verantwortlich, dass berechtigte IT-Service-Nutzer die korrekten Zugriffrechte auf IT-Services, Daten und andere Vermögenswerte erhalten, um die Vertraulichkeit, Verfügbarkeit und Integrität dieser zu gewährleisten.

2.5.5 Continual Service Improvement

Die Phase *Continual Service Improvement* [Offi11e] ist für die Verwaltung von Verbesserungen in IT-Services und in den IT-Service-Management-Prozessen verantwortlich. Es werden Verbesserungen über den gesamten IT-Service-Lebenszyklus betrachtet, um die Effektivität, Effizienz und Wirtschaftlichkeit der IT-Infrastruktur, den Prozessen, sowie den angebotenen IT-Services zu verbessern. Alle Aktivitäten in der Phase *Continual Service Improvement* werden im Prozess *Seven-Step Improvement* beschrieben.

2.6 Change Management

Wie in Kapitel 2.5.3 bereits kurz beschrieben, ist der Gegenstand des Prozesses *Change Management* die Planung, Steuerung und Kontrolle von Changes [Offi11a], um den laufenden Betrieb der IT-Services möglichst ohne Störungen fortführen zu können [Leim12]. In ITIL wird unter dem Begriff Change das Hinzufügen, die Veränderung oder die Entfernung eines IT-Services verstanden [BoJK08], worunter somit auch die Wartung eines IT-Services fällt. Die Wartung eines IT-Services wird in dieser Arbeit angelehnt an den ISO/IEC Standard 14764-2006 [Inte06] wie folgt definiert:

Definition 2.10: IT-Service-Wartung

IT-Service-Wartung ist die Veränderung eines sich in Betrieb befindenden IT-Services, um

- diesen an sich ändernde oder geänderte Umgebungsbedingungen anzupassen,
- enthaltene Fehler zu korrigieren, oder
- Optimierungen am IT-Service vorzunehmen.

■

Die Wartung eines IT-Services stellt somit einen Change dar, ist jedoch nur eine Untermenge möglicher Changes, da beispielsweise das erstmalige Hinzufügen eines IT-Services in die Produktionsumgebung keine Wartung ist.

Wartungen können sowohl einen proaktiven als auch einen reaktiven Grund haben. Ein proaktiver Grund für die Änderung eines IT-Services liegt beispielsweise vor, wenn dadurch die Kosten zur Erbringung des IT-Services reduziert werden können oder ein latenter Fehler im IT-Service entdeckt wurde, und dieser ausgebessert wird, bevor er zu einer Störung des IT-Services führen kann. Ein reaktiver Grund liegt vor, wenn beispielsweise ein Software-Fehler bereits zu Störungen des IT-Services führt, und dieser Fehler mit einer neuen Software-Version behoben werden soll [Inte06].

In der betrieblichen Praxis hat sich für die Durchführung des Change Managements folgende grundlegende Methodik etabliert (vgl. [Cisc08, CMAS09, KHWW04, Offi11a]):

1. Zunächst erfolgt eine formale Beantragung einer geplanten Wartung eines IT-Services in Form eines RFCs, in dem beschrieben wird, **was** geändert werden soll, z.B. ob eine neue Softwareversion installiert werden soll, oder die Konfiguration einer Software geändert werden soll, und bis wann die Änderung vorgenommen werden soll.
2. Anschließend wird die im RFC dokumentierte Anpassung evaluiert, und die Auswirkungen bei Umsetzung der gewünschten Änderungen auf den zu ändernden IT-Service und Abhängigkeiten zu weiteren IT-Services analysiert. Der RFC wird auf Basis der Analyseergebnisse entweder abgelehnt, und die geplante Änderung somit nicht durchgeführt, oder der Antrag wird genehmigt.
3. Wenn der RFC genehmigt wird, wird eine detaillierte Ablaufplanung zur Änderung des IT-Services erstellt. Dazu wird ein Change Plan erstellt, in dem beschrieben wird, **wie** die Änderung durchzuführen ist. Dieser Change Plan enthält alle durchzuführenden Aktivitäten, die verantwortlichen Aufgabenträger je Aktivität, zeitliche Vorgaben über die Dauer und späteste Fertigstellung von Aktivitäten, sowie Eskalationsbeschreibungen für Fehlersituationen. Aktivitäten auf einem hohen Abstraktionslevel können beispielsweise sein: *Installiere Datenbank-Server, Konfiguriere Datenbank-Server, Erstelle Datenbank, Installiere Web-Server, Konfiguriere Web-Server, Deploye Web-Anwendung*. In Kapitel 1.2.1 werden Arbeiten vorgestellt, in denen die Erstellung von Change Plänen beschrieben wird.
4. Der Change Plan wird von einem ausgewählten fachkundigen Personenkreis überprüft und für die Durchführung von Tests freigegeben. Während der Tests wird der Change Plan bei Bedarf aktualisiert.

5. Nach erfolgreicher Durchführung der Tests, wird der Change für die Durchführung in der Produktionsumgebung zeitlich geplant und für die Durchführung in diesem Zeitraum freigegeben. Innerhalb dieses Zeitraums wird dann der Change Plan final ausgeführt.
6. Nach der Durchführung des Changes wird dieser hinsichtlich seines Ziels und der Art der Durchführung, ob es zu eventuellen Komplikationen oder Seiteneffekten auf andere IT-Services gekommen ist, bewertet.

Während der Durchführung einer Wartung kann es zu verschiedenen Fehlern kommen. Werden diese Fehler nicht vor Abschluss der Wartung erkannt und behoben, kann es zu einer Störung des betroffenen IT-Services führen.

2.7 Zusammenfassung des Kapitels

In diesem Kapitel wurden die Grundlagen des IT-Service-Managements beschrieben. Dazu wurde zunächst definiert was ein Service bzw. IT-Service ist und welche Rollen an einem IT-Service beteiligt sind. Aufbauend auf diesen grundlegenden Definitionen wurden typische Prozesse zum Management von IT-Services anhand des De-facto-Standards ITIL beschrieben. Die Planung, Kontrolle und Steuerung von IT-Service-Wartungen ist Bestandteil des Prozesses Change Management. Aufgrund dessen wurden die typischen Durchführungsschritte des Change Managements in der betrieblichen Praxis detailliert beschrieben. ITIL lässt dabei die genaue Ausgestaltung der einzelnen Durchführungsschritte offen. Um Durchführungsfehler während der IT-Service-Wartung zu vermeiden, empfiehlt ITIL ausgiebiges Testen vor der Wartungsdurchführung. Nach der Durchführung einer IT-Service-Wartung soll analysiert werden, ob es zu irgendwelchen Komplikationen oder Fehlern während der Wartung kam, was die Gründe dafür waren, und wie diese zukünftig vermieden werden können. Eine Empfehlung, wie jedoch Fehler während der Durchführung der IT-Service-Wartung erkannt werden können, wird nicht beschrieben. Diese Lücke soll in dieser Arbeit geschlossen werden. Im nachfolgenden Kapitel werden dafür zunächst die potenziell möglichen Fehler während einer IT-Service-Wartung und die Grundsätze zur Fehlererkennung beschrieben.

3 Fehler und Fehlererkennung

Damit eine Methode zur Entdeckung von Wartungsfehlern entwickelt werden kann, muss zunächst analysiert werden, was Wartungsfehler sind, wie sich solche äußern und welche Arten von Wartungsfehlern auftreten können. In diesem Kapitel werden dazu zunächst die Begriffe Fehler und Störungen im Kontext von IT-Services definiert und anschließend eine generelle Klassifizierung von Fehlerarten vorgenommen. Daraufhin werden, auf Basis einer Literaturrrecherche, empirisch nachgewiesene Wartungsfehler vorgestellt und mit Beispielen verdeutlicht. Das Kapitel schließt mit einer Erläuterung allgemeiner Prinzipien zur Fehlererkennung und einer Veranschaulichung, wie Wartungsfehler mit der in dieser Arbeit entwickelten Methode erkannt werden können.

3.1 Begriffsklärung

In ITIL werden Störungen eines IT-Services als „Incidents“ bezeichnet, und als eine ungeplante Unterbrechung eines IT-Services, oder als Minderung der Qualität eines IT-Services, beschrieben. Der Ausfall einer einzelnen Komponente eines maschinellen Aufgabenträgers, beispielsweise einer Festplatte, gilt ebenso als Störung, auch wenn der IT-Service für den IT-Service-Abnehmer weiterhin zur Verfügung steht [Offi11d]. In dieser Arbeit wird der Begriff der Störung eines IT-Services, orientiert an der verbreiteten Taxonomie von Laprie [Lapr95], wie folgt definiert:

> **Definition 3.1: IT-Service-Störung**
>
> Eine IT-Service-Störung bezeichnet eine ungeplante Abweichung des ausgeführten IT-Services vom Versprechen des IT-Service-Anbieters an den IT-Service-Abnehmer.
>
> ■

In Definition 3.1 wird auf das Versprechen des IT-Service-Anbieters gegenüber dem IT-Service-Abnehmer Bezug genommen (siehe Definition 2.1: Service, Dienstleistung), welches gemäß ITIL in der Phase Service Design in Form einer Service-Spezifikation im Service Katalog festgehalten wird. Je nach Abweichung von der Spezifikation des IT-Services, kann sich eine IT-Service-Störung[1] unterschiedlich stark äußern, beispielsweise kann lediglich die

[1] Aus Gründen der Lesbarkeit wird im weiteren Verlauf der Arbeit statt IT-Service-Störung der Begriff Störung verwendet.

Performanz beeinträchtigt sein, es können jedoch auch grundsätzliche Funktionalitäten ausfallen, oder der ganze IT-Service kann für den IT-Service-Abnehmer nicht erreichbar sein. Die Unterbrechung eines IT-Services stellt keine Störung da, solange diese geplant und dem IT-Service-Anbieter beispielsweise im Rahmen eines SLAs mitgeteilt wurde. Im Gegensatz zu ITIL wird der Ausfall einer einzelnen Komponente eines IT-Services, der zu keiner Abweichung von der Spezifikation des IT-Services führt, in dieser Arbeit nicht als Störung betrachtet. Ein solch unzulässiger Zustand eines IT-Services wird gemäß der Taxonomie von Laprie [Lapr95] als „Fehlerzustand“[2] bezeichnet. Die Ursache von einem Fehlerzustand ist ein Fehler. Die Transition von einem ursprünglich korrekten Zustand in einen Fehlerzustand wird als Ausfall [ALRL04] bezeichnet. Eine IT-Service-Störung kann wiederum der Grund für weitere Fehler sein [DeKL98].

Definition 3.2: Fehlerzustand

Ein Fehlerzustand bezeichnet den unzulässigen Zustand des IT-Services, der zu einer IT-Service-Störung führen kann.

■

Definition 3.3: Fehler

Ein Fehler ist die Ursache eines Fehlerzustands und kann intern im IT-Service verankert sein, oder von außen in den IT-Service eingebracht werden.

■

Abbildung 3.1 verdeutlicht den Zusammenhang zwischen einem Fehler, einem Fehlerzustand und einer Störung.

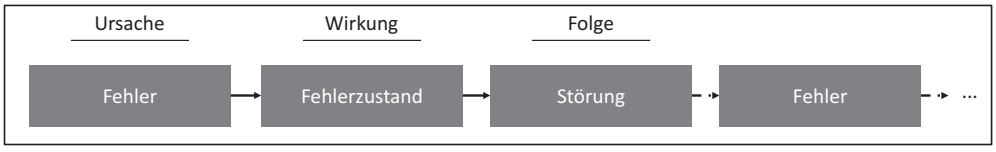


Abbildung 3.1: Zusammenhang Fehler, Fehlerzustand und Störung (angelehnt an [DeKL98])

[2] In der zitierten Arbeit wird der englische Begriff „error“ verwendet. Im Rahmen dieser Arbeit wird der deutsche Begriff „Fehlerzustand“ verwendet.

3.2 Klassifizierung von Fehlern

Nachdem die grundlegenden Begriffe definiert sind, wird nachfolgend eine Klassifizierung von möglichen Fehlern vorgenommen, um anschließend eine Eingrenzung auf die Fehler vorzunehmen, die durch die in dieser Arbeit vorgestellten Methode entdeckt werden können. Avizienis et al. [ALRL04] klassifizieren Fehler nach folgenden acht Dimensionen:

- Betrachtungseinheit
  Die Betrachtungseinheit beschreibt, ob ein Fehler in der Software oder in der Hardware des Anwendungssystems auftritt. In dieser Arbeit werden nur Fehler berücksichtigt, die die Wartung der Software betreffen.
- Phänomenologischer Grund
  In dieser Dimension werden Fehler danach klassifiziert, ob sie aufgrund gegebener Naturgesetze auftreten oder durch den Menschen verursacht sind. Fehler, die aufgrund von Naturgesetzen auftreten, können bei einem IT-Service nur bei der eingesetzten Hardware vorkommen. Naturgesetze haben aufgrund des immateriellen Charakters von Software keine Auswirkungen auf diese. Menschen hingegen können sowohl Fehler in die Hardware einbringen, beispielsweise durch den falschen Einbau eines Kondensators, als auch in die Software, beispielsweise durch Entwurfs,- Programmier- oder Wartungsfehler [HaKo13]. Eine weitere Klassifizierung von Fehlern, die durch den Menschen verursacht werden, kann anhand der an einem IT-Service beteiligten Rollen vorgenommen werden. Dem IT-Service-Anbieter können Fehler beim Aufbau, der Bereitstellung und der Wartung des IT-Services entstehen. Dem IT-Service-Abnehmer können Bedienfehler bei der Nutzung des IT-Services unterlaufen. Zusätzlich können Fehler durch Angriffe externer Dritter entstehen.
- IT-Service-Phase
  Diese Dimension beschreibt in welcher Phase des IT-Services ein Fehler auftritt. Ein Fehler kann in der Entwicklung und Bereitstellung des IT-Services durch den IT-Service-Anbieter, oder während der Nutzung eines IT-Services durch den IT-Service-Abnehmer stattfinden. Fehler während der Nutzung können Bedienfehler oder Fehler durch Hackerangriffe sein. Die für diese Arbeit relevanten Wartungsfehler treten im Betrieb eines IT-Services auf, wenn während des Betriebs eine Wartung durchgeführt wird. In ITIL entspricht das der Phase Service Transition.

- IT-Service-Sicht
  Diese Dimension beschreibt, ob Fehler innerhalb der Grenzen des IT-Services auftreten, beispielsweise durch Alterung der Hardware, oder von außen in den IT-Service eingebracht werden. Fehler, die von außen in den IT-Service eingebracht werden, sind beispielsweise Konfigurationsfehler während einer Wartung.
- Ziel
  Das Ziel beschreibt, ob ein Fehler bewusst in den IT-Service eingeschleust wurde, um eine Störung zu provozieren, oder der Fehler ohne böse Absicht eingespielt wurde. In dieser Arbeit werden nur Wartungsfehler untersucht, die nicht zum Ziel haben, eine Störung zu verursachen.
- Kenntnis
  In dieser Dimension werden Fehler danach klassifiziert, ob sie bewusst aufgrund einer leichtsinnigen bzw. risikoreichen Entscheidung eingespielt wurden, oder die Fehler unbewusst eingespielt wurden.
- Fähigkeit
  Fehler können danach unterteilt werden, ob sie unbeabsichtigt eingespielt wurden, oder die Fehler aufgrund von Inkompetenz oder nicht ausreichender Kompetenz der am IT-Service beteiligten Menschen verursacht wurden.
- Persistenz
  In der Dimension Persistenz werden Fehler danach klassifiziert, ob sie nur in einem beschränkten Zeitrahmen im IT-Service auftreten, beispielsweise nur beim Start eines IT-Services, oder angenommen wird, dass die Fehler während der gesamten IT-Service-Nutzungsdauer auftreten.

Zusätzlich zu diesen acht Dimensionen wird für die Fehler-Eingrenzung folgende Dimension von Halang und Konakovsky berücksichtigt [HaKo13]:

- Entstehungsprozess
  Menschliche Fehler können zudem hinsichtlich ihres Entstehungsprozesses in Spezifikations- und Umsetzungsfehler unterschieden werden. Spezifikationsfehler sind Fehler, bei denen die beabsichtigte Funktion nicht der spezifizierten Funktion entspricht. Im Falle der in dieser Arbeit untersuchten Wartungsfehler, wäre ein Spezifikationsfehler, wenn beispielsweise der falsche zu wartende Server in einem Change Plan definiert wird. Umsetzungsfehler sind Fehler, die bei der Umsetzung einer Spezifikation entstehen, beispielsweise, wenn zwar der korrekte zu wartende Server spezifiziert ist, jedoch ein anderer Server gewartet wird.

Die in dieser Arbeit entwickelte Methode ermöglicht die Entdeckung von Softwarewartungsfehlern bei IT-Services, die bei der Umsetzung des Change Plans durch den IT-Service-Anbieter auftreten können. Die Fehler werden daher von außen und ohne Absicht in den IT-Service eingebracht. Die Methode eignet sich zur Fehlerentdeckung unabhängig davon, ob die Fehler aufgrund mangelnder Fähigkeiten der durchführenden Personen, und mit oder ohne Kenntnisnahme des Fehlers eingespielt werden. Die Methode ist zudem unabhängig von der Persistenz des Fehlers im IT-Service anwendbar. Abbildung 3.2 veranschaulicht die mit der Methode erkennbaren Fehler.

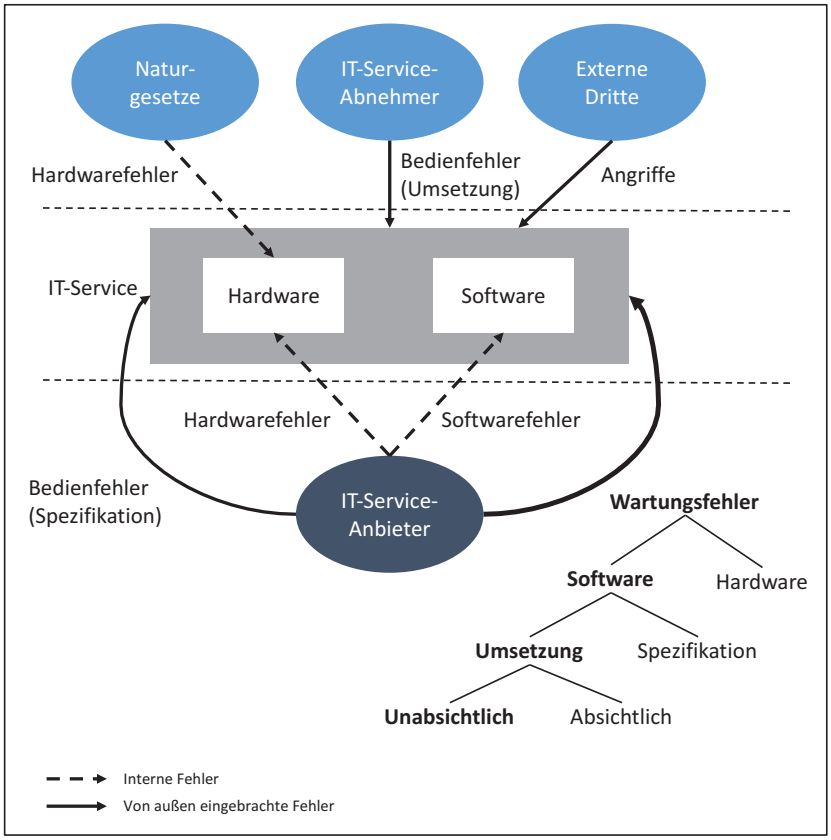


Abbildung 3.2: Eingrenzung der in dieser Arbeit relevanten Fehlerklasse

In diesem Kapitel wurden Wartungsfehler allgemein klassifiziert und für den Untersuchungsgegenstand dieser Arbeit auf unabsichtliche Fehler bei der Umsetzung der Softwarewartung eingegrenzt. Im nachfolgenden Kapitel werden diese auf Basis empirischer Ergebnisse weiter klassifiziert und anhand von Beispielen typische Fehler aufgezeigt.

Das Ziel ist es nicht alle theoretisch möglichen Fehler zu beschreiben, sondern typische Fehler, wie sie tatsächlich in der betrieblichen Praxis vorkommen und nachgewiesen werden konnten, vorzustellen.

3.3 Wartungsfehler

Oppenheimer et al. [OpGP03] unterteilen Wartungsfehler von IT-Service-Störungen in Ablauf- und Konfigurationsfehler. Diese Aufteilung findet sich auch in weiteren Arbeiten über Wartungsfehler von IT-Services, wie beispielsweise [DuNa09, NOBM04, ONBB06, PeNa05], und wird auch für diese Arbeit verwendet.

Pertet und Narasimhan [PeNa05] haben verschiedene Fallstudien und Fehlerberichte hinsichtlich der Gründe für IT-Service-Störungen in Web-Anwendungen untersucht und Wartungsfehler in Konfigurations-, Ablauffehler und unglückliche Unfälle unterteilt. Unglückliche Unfälle betreffen die Wartung von Hardware, wenn diese beispielsweise aus Versehen fallen gelassen wird und dadurch Schaden nimmt. Da sich diese Wartungsfehler auf Hardware beziehen, werden sie in dieser Arbeit nicht weiter betrachtet. Ablauffehler entstehen, wenn beispielsweise eine Aktivität bei der Wartung vergessen wurde, oder eine falsche Aktivität durchgeführt wurde [PeNa05]. Falsche Aktivitäten sind beispielsweise das Einspielen des falsches Backups [PeNa05], oder die Installation einer 64bit Anwendung auf einem 32bit Betriebssystem, statt eines 64bit Betriebssystems [ZhLi11]. Konfigurationsfehler beschreiben Fehler, die aufgrund unzulässiger Konfigurationsparameter entstehen, werden jedoch von Pertet und Narisimhan nicht weiter klassifiziert [PeNa05].

Auf Basis der empirischen Ergebnisse aus [DuNa09, NOBM04, ONBB06, PeNa05] werden Ablauffehler weiter unterteilt in die Kategorien „Auftreten“ und „Reihenfolge“. Der Kategorie „Auftreten“ werden Fehler zugeordnet, die durch das Vergessen einer Aktivität oder der Ausführung einer unnötigen Ausführung entstehen. Es sei angemerkt, dass das Vergessen einer Aktivität nur dann ein Wartungsfehler im Sinne des Untersuchungsgegenstands dieser Arbeit ist, wenn die Aktivität im Change Plan spezifiziert war. War diese Aktivität nicht spezifiziert, handelt es sich um einen Spezifikationsfehler und nicht um einen Umsetzungsfehler. Die Durchführung einer unnötigen Aktivität liegt vor, wenn die Aktivität nicht im Change Plan spezifiziert ist.

Der Kategorie „Reihenfolge“ werden Fehler zugeordnet, die durch die falsche Durchführung einer Aktivität entstehen, oder eine Aktivität in der falschen Reihenfolge durchgeführt wurde. Eine Aktivität wird falsch durchgeführt, wenn beispielsweise die falsche Version einer Software installiert wird. Eine Aktivität wird in der falschen Reihenfolge durchgeführt,

wenn beispielsweise ein Server vor seiner Konfiguration, anstatt nach seiner Konfiguration neu gestartet wird. In Tabelle 3.1 werden die genannten Ablauffehlerarten mit realen Beispielen aufgelistet.

Tabelle 3.1: Arten von Ablauffehlern

Fehler-kategorie	Fehler	Beispiel	Referenz
Auftreten	Aktivität vergessen	Vergessen, den Web Server neu zu starten.	[PeNa05]
	Unnötige Aktivität durchgeführt	Server unnötigerweise neu gestartet.	[NOBM04]
Reihenfolge	Aktivität falsch durchgeführt	Falsches Backup eingespielt.	[PeNa05]
	Aktivität in der falschen Reihenfolge durchgeführt	Zwei Server gleichzeitig herunterfahren, anstatt diese nacheinander zu warten.	[NOBM04]

Yin et al. [YMZZ11] haben 546 real stattgefundene Konfigurationsfehler in einem kommerziell genutzten System und vier weit verbreiteten Open-Source-Systemen untersucht und diese in legale und illegale Konfigurationsfehler klassifiziert. Illegale Fehler sind Fehler, die vom zu konfigurierenden IT System nicht akzeptiert bzw. verstanden werden, beispielsweise aufgrund einer falschen Rechtschreibung des Parameters. Illegale Fehler können mithilfe von Konfigurationsregeln entdeckt werden. Legale Fehler sind Parameter, die vom System an sich akzeptiert werden, jedoch beispielsweise zu Performance Einbußen führen können. Legale Fehler lassen sich durch das Testen der Konfigurationseinstellungen aufdecken und entsprechen gemäß der Fehlerklassifizierung in Kapitel 3.2 Spezifikationsfehlern, da diese aufgrund einer falschen Spezifikation umgesetzt werden. Diese legalen Fehler werden daher im Rahmen dieser Arbeit nicht weiter berücksichtigt.

Illegale Fehler lassen sich weiter unterteilen in Format- und Wert-Fehler. Format-Fehler können entweder lexikalischer oder syntaktischer Natur sein. Lexikalische Fehler sind Fehler, die gegen die erlaubte Grammatik eines Parameters verstoßen. Syntaktische Fehler verstoßen gegen strukturelle Vorgaben. Wert-Fehler werden weiter klassifiziert in Typos, Wert-Inkonsistenzen, und Umgebungs-Inkonsistenzen. In Tabelle 3.2 werden die verschiedenen Konfigurationsfehler mit Beispielen aufgeführt.

Tabelle 3.2: Arten von Konfigurationsfehlern (in Anlehnung an [YMZZ11])

Fehler-kategorie	Fehler	Beispiel	Fehlerbeschreibung
Format	Lexikalischer Fehler	InitiatorName: iqn:DEV_domain	Bei diesem Parameter sind nur Kleinbuchstaben erlaubt. Die Eingabe DEV kann somit nicht vom Programm interpretiert werden.
	Syntaktischer Fehler	extension = mysql.so ..... extension = recode.so	Es besteht eine Abhängigkeit zwischen den beiden Werten mysql.so und recode.so. Damit die Anwendung korrekt arbeitet, muss in der Konfigurationsdatei recode.so vor mysql.so gesetzt werden.
	Typo	extension = recdoe.so extension = mysql.so	Der Wert recdoe.so muss eigentlich recode.so heißen.
Wert	Wert-Inkonsistenz	log_output = "Table" log = query.log	log_output gibt an, dass die Logs in eine Datenbanktabelle geschrieben werden sollen. Mit der Angabe log=query.log wird jedoch auf eine Log-Datei referenziert.
	Umgebungs-Inkonsistenz	datadir = /some/old/path	Der angegebene Pfad ist veraltet und enthält die relevanten Dateien nicht mehr oder der Pfad existiert überhaupt nicht (mehr).

Abbildung 3.3 stellt die verschiedenen Typen von Wartungsfehlern übersichtlich dar.

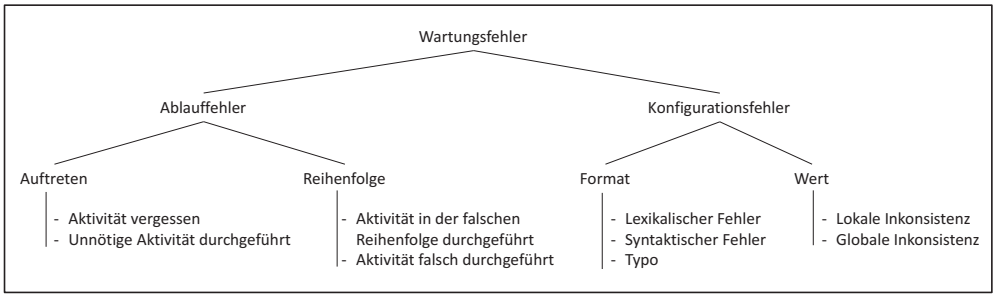


Abbildung 3.3: Klassifizierung von Wartungsfehlern

3.4 Allgemeines Prinzip der Fehlererkennung

Nachdem in Kapitel 3.3 die verschiedenen Fehler, die bei der Wartung eines IT-Services auftreten können, beschrieben wurden, wird nun das allgemeine Prinzip der Fehlererkennung vorgestellt. Das allgemeine Prinzip der Fehlererkennung beschreibt, wie Fehler, unabhängig von der Betrachtungseinheit, z.B. Hardware, Software, Spezifikationen, oder Fehler im menschlichen Denken, erkannt werden können. Es ist wie folgt formuliert [Kona88]:

„In einem beliebigen Verfahren zur Erkennung von Fehlern werden mindestens zwei Werte auf die Erfüllung der zwischen diesen Werten vorgegebenen Zusammenhänge geprüft. Unzulässige Abweichungen von diesen Zusammenhängen werden als Fehler interpretiert“ [Kona88]. Abbildung 3.4 stellt das allgemeine Prinzip der Fehlererkennung schematisch dar.

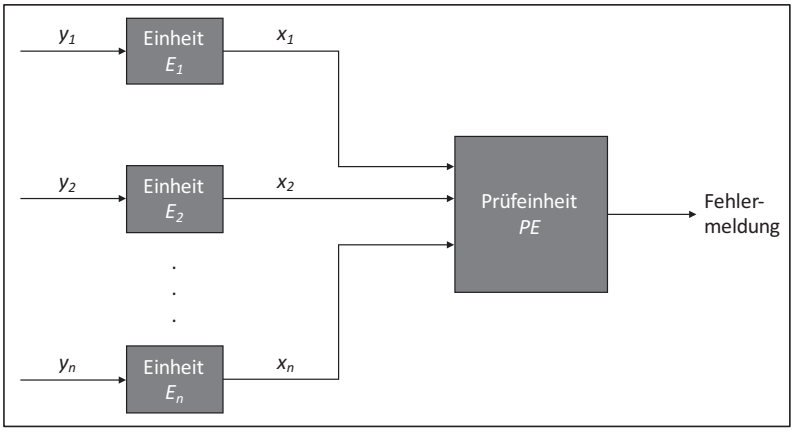


Abbildung 3.4: Allgemeines Prinzip der Fehlererkennung (angelehnt an [HaKo13])

Um Fehler erkennen zu können, werden die Werte x_1, x_2, ..., x_n auf einen oder mehrere vorbestimmte Zusammenhänge, durch die dafür vorgesehene Prüfeinheit PE, überprüft. Werden die vorbestimmten Zusammenhänge nicht erfüllt, meldet die Prüfeinheit PE einen Fehler. Die Werte x_1, x_2, x_3, ..., x_n werden durch die Einheiten E_1, E_2, E_3, ..., E_n mit den Eingangswerten y_1, y_2, y_3, ..., y_n gewonnen [HaKo13].

Die Prüfung auf eine Abweichung kann dabei durch eine Plausibilitätsprüfung oder einen Vergleich erfolgen, und wird nachfolgend jeweils anhand eines Beispiels, wie von Halang und Konakovsky [HaKo13] beschrieben, erläutert.

3.4.1 Fehlererkennung durch einen Vergleich

Bei einer Fehlererkennung durch einen Vergleich werden die Ausgangswerte x_1 und x_2 durch zwei identische, parallel betriebene, Messeinrichtungen E_1 und E_2 miteinander verglichen. Voraussetzung ist, dass E_1 und E_2 denselben Eingangswert y verarbeiten. Die Ausgangswerte x_1 und x_2 sollten in diesem Fall übereinstimmen: $x_1 = x_2$. Wenn dies nicht der Fall ist, arbeitet eine der beiden Einheiten fehlerhaft. Ein Anwendungsfall für diese Art der Fehlererkennung ist beispielsweise der Vergleich von zwei Thermometern, die dieselbe Raumtemperatur messen. Für die Fehlererkennung bei der Durchführung von IT-Service-Wartungen ist diese Art der Fehlererkennung ungeeignet. Abbildung 3.5 stellt die beispielhafte Fehlererkennung durch einen Vergleich bei zwei Thermometern dar.

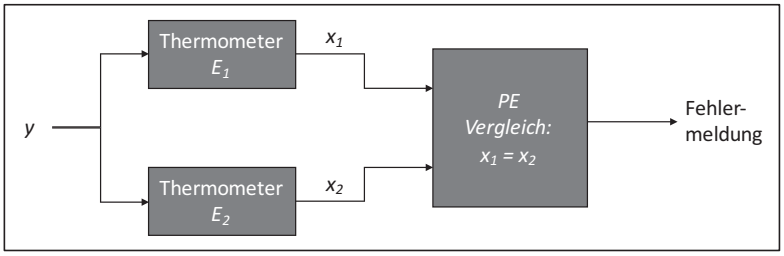


Abbildung 3.5: Fehlererkennung durch einen Vergleich (angelehnt an [HaKo13])

3.4.2 Fehlererkennung durch eine Plausibilitätsprüfung

Eine andere Art der Fehlererkennung ist die Fehlererkennung durch eine Plausibilitätsprüfung. Im Gegensatz zur Fehlererkennung durch einen Vergleich werden bei dieser Art der Fehlererkennung keine zwei identischen Einheiten E_1 und E_2 benötigt. Zur Veranschaulichung der Fehlererkennung durch eine Plausibilitätsprüfung soll folgendes Beispiel dienen:

x_1 ist die gemessene Ausführungszeit einer Aktivität *A* eines IT-Services,

x_2 ist die spezifizierte minimale Zeit, die für die Ausführung der Aktivität *A* benötigt wird,

x_3 ist die spezifizierte maximale Zeit, die für die Ausführung der Aktivität *A* benötigt wird.

Bei jeder Ausführung der Aktivität *A* werden die Ausgangswerte x_1, x_2 und x_3 auf folgenden Zusammenhang durch die Prüfungseinheit überprüft:

$$x_2 <= x_1 <= x_3$$

Gilt dieser Zusammenhang bei der Ausführung von Aktivität *A* nicht, liegt ein Fehler vor. Die minimalen und maximalen Werte können dabei zusätzlich von einem Parameter *y* abhängig sein, beispielsweise dem mit dem Kunden vereinbarten SLA und damit der vereinbarten Antwortzeit von Aktivität *A*. Die Überprüfung solcher Zusammenhänge wird als Plausibilitätsprüfung bezeichnet [HaKo13]. Abbildung 3.6 stellt die beispielhafte Fehlererkennung durch eine Plausibilitätsprüfung dar.

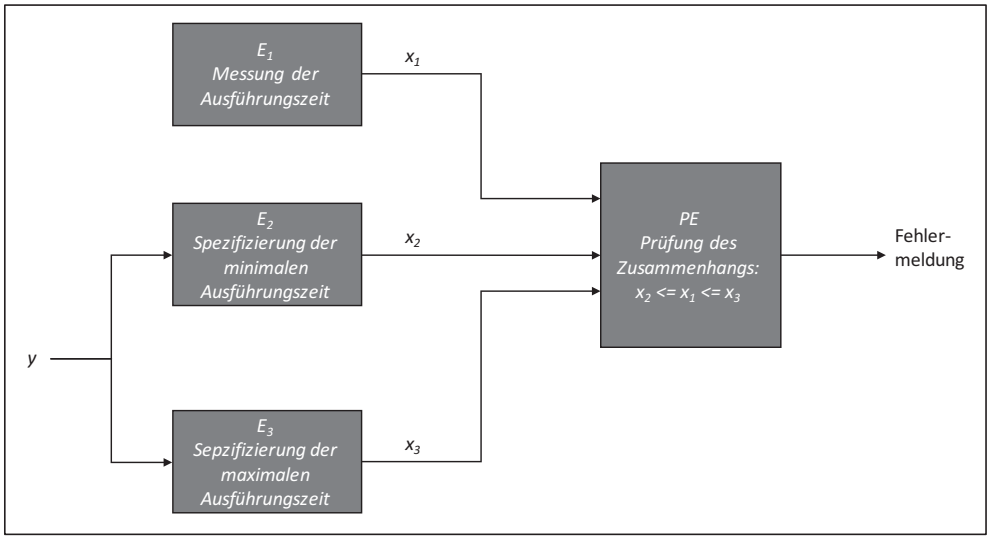


Abbildung 3.6: Fehlererkennung durch eine Plausibilitätsprüfung (angelehnt an [HaKo13])

3.5 Fehlererkennung bei der Durchführung von IT-Service-Wartungen

Nachdem das allgemeine Prinzip der Fehlererkennung beschrieben wurde, wird an dieser Stelle dokumentiert, wie mithilfe der in dieser Arbeit entwickelten Methode, die Erkennung von Fehlern bei der Durchführung von IT-Service-Wartungen erfolgt.

Gemäß dem allgemeinen Prinzip der Fehlererkennung müssen mindestens zwei Werte auf einen festgelegten Zusammenhang hin überprüft werden. Im Falle der IT-Service-Wartung werden hierzu die Ausgangswerte, der IT-Service-Komponenten, und die Werte der eingesetzten Software zur Überwachung der IT-Service-Komponenten, auf Plausibilität zum a priori erstellten Ablaufmodell überprüft.

Die Ausgangswerte x_1 bis x_n seien

- Ereignisse[3], die eine Wartungsaktivität einer IT-Service-Komponente E_1, E_2, ..., E_n repräsentieren. Ein solches Ereignis kann beispielsweise das Stoppen eines Servers oder die Veränderung eines Konfigurationsparameters sein.

Der Ausgangswert $ü$ sei

- der ermittelte Zustand (online/ offline) einer IT-Service-Komponente E_1, E_2, ..., E_n, der mithilfe der Überwachungsanwendung $E_ü$ gemessen wird.

Der Ausgangswert p sei

- eine Pattern-Instanz[4], die aus dem Ablaufmodell instanziiert wird, beispielsweise dass die Aktivität x_2 nach der Aktivität x_1 ausgeführt werden muss.

Diese Ausgangswerte werden durch die Prüfungseinheit *PE* auf Plausibilität zueinander geprüft. Die Überprüfung der Plausibilität erfolgt dabei anhand von Anti-Pattern, d.h. es wird eine Fehlermeldung ausgegeben, wenn die Ausgangswerten x_1, x_2, ..., x_n und $x_ü$ nicht mit einer *Pattern-Instanz* übereinstimmen, zum Beispiel, wenn x_2 vor anstatt nach x_1 ausgeführt wird. Abbildung 3.7 veranschaulicht die in dieser Arbeit eingesetzte Methode zur Fehlererkennung.

[3] Der Begriff Ereignis wird in Kapitel 5.3 im Detail beschrieben.
[4] Der Begriff Pattern wird in Kapitel 5.1 näher beschrieben und definiert.

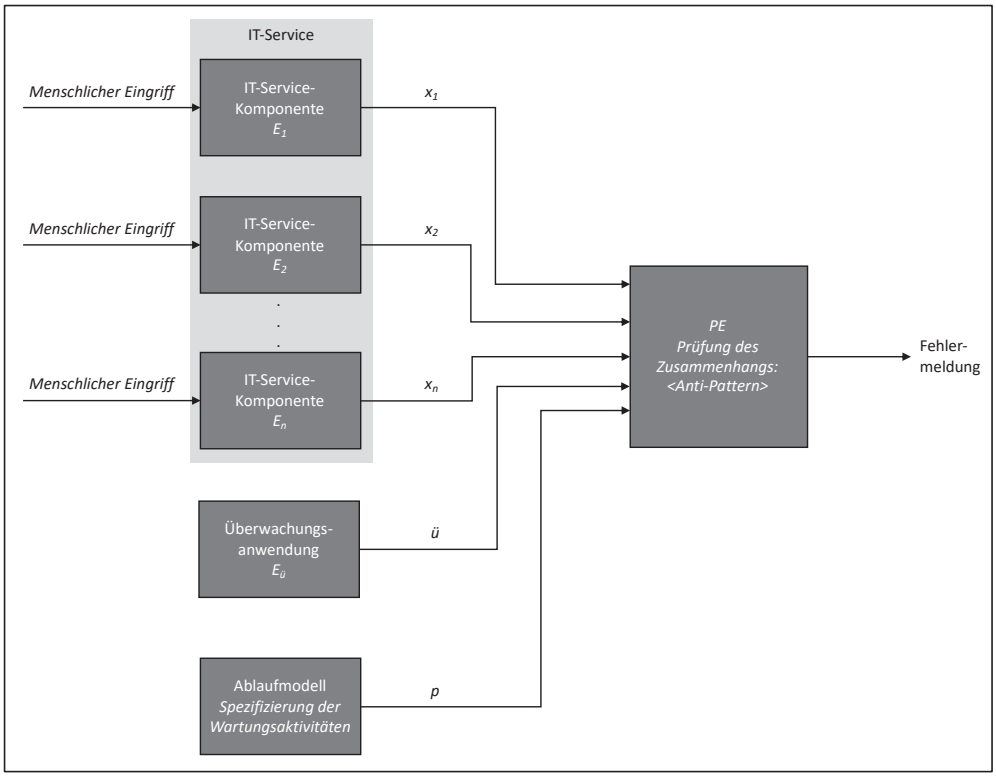


Abbildung 3.7: Methode zur Fehlererkennung bei IT-Service-Wartungen

3.6 Zusammenfassung des Kapitels

In diesem Kapitel wurde der Untersuchungsgegenstand des Wartungsfehlers analysiert. Hierfür wurden zunächst die Begriffe Fehler, Fehlerzustand und IT-Service-Störung definiert und voneinander abgegrenzt. Anschließend wurden generell mögliche Fehler beim Betrieb von IT-Services nach verschiedenen Dimensionen klassifiziert, um so eine Eingrenzung der für diese Arbeit relevanten Wartungsfehler vorzunehmen und die möglichen Fehler von anderen möglichen Fehlerursachen abzugrenzen. Nach der Eingrenzung von Wartungsfehlern hinsichtlich der verschiedenen Dimensionen eines Fehlers im Kontext von IT-Services, wurden Wartungsfehler selbst weiter unterteilt in Konfigurations- und Ablauffehler. Ablauffehler können unterschieden werden in Fehler hinsichtlich der Reihenfolge von Aktivitäten, z.B. dass eine Aktivität vor anstatt nach einer anderen Aktivität ausgeführt wurde, oder Fehler hinsichtlich des Auftretens einer Aktivität, z.B. wenn vergessen wurde eine Aktivität

auszuführen. Konfigurationsfehler können nach Format- und inhaltlichen Fehler unterschieden werden. Format-Fehler sind z.B. syntaktische Fehler bei der Konfiguration, wohingegen inhaltliche Fehler zu Inkonsistenzen in den konfigurierten IT-Service-Komponenten führen. Sowohl Konfigurations- als auch Ablauffehler sollen von der in dieser Arbeit entwickelten Methode erkannt werden. Zur Veranschaulichung der Funktionsweise der Methode wurde in diesem Kapitel zudem das allgemeine Prinzip der Fehlererkennung vorgestellt und auf den Kontext dieser Arbeit zur Erkennung von Wartungsfehlern übertragen. Für die in dieser Arbeit vorgestellte Methode wird zur Fehlererkennung ein Ablaufmodell der IT-Service-Wartung benötigt. Die Grundlagen der Modellierung und die Erstellung eines Ablaufplans zur Beschreibung von IT-Service-Wartungen werden im nachfolgenden Kapitel vorgestellt.

4 Methoden zur Modellierung von IT-Service-Wartungen

Im Rahmen dieser Arbeit wird die Modellierung von IT-Service-Wartungen zum Zweck der Überwachung der tatsächlichen Wartungsdurchführung verwendet. In Kapitel 3.3 wurden Wartungsfehler in Ablauf- und Konfigurationsfehler unterteilt. Demnach ist für eine Erkennung von Fehlern bei der Durchführung von Wartungen sowohl eine Modellierung des Wartungsablaufs, als auch der zu wartenden Objekte, in diesem Fall der zu wartenden IT-Service-Komponenten, notwendig. In diesem Kapitel wird zunächst eine Einführung in die Modellierung allgemein gegeben, um anschließend verschiedene Methoden zur Modellierung von IT-Service-Wartungen vorzustellen.

4.1 Einführung in die Modellierung

Gemäß Oberweis [Ober96, S.19] wird unter Modellierung die Beschreibung von bestimmten Sachverhalten, z.B. von Abläufen, IT-Systemen, Wetterphänomenen, etc. verstanden. Zur Beschreibung wird eine Beschreibungssprache, das Beschreibungsmodell, verwendet. Das Ergebnis der Modellierung bzw. die konkrete Beschreibung eines Sachverhalts wird als Schema bezeichnet und kann bei der Modellierung von Abläufen, wie es IT-Service-Wartungen sind, zu unterschiedlichen Zwecken verwendet werden [Ober96, S.19ff.]:

- Zu Dokumentationszwecken
  Die Modellierung kann eingesetzt werden, um Abläufe, z.B. den Ablauf einer Wartung, zu dokumentieren und als Handlungsanweisung für die durchführenden Personen verwendet werden.
- Zur Analyse und Reorganisation von Abläufen
  Analysen können sich sowohl auf syntaktische, als auch auf inhaltliche Aspekte beziehen, um beispielsweise die Widerspruchsfreiheit oder Durchlaufzeit einer IT-Service-Wartung zu untersuchen. Für ein Beispiel zur Analyse der Widerspruchsfreiheit bei der Wartung einer Cloud-Anwendung siehe [BCSW15].
- Zu Entwurfszwecken
  In bestimmten Fällen können Ablaufbeschreibungen zur Generierung von Anwendungen eingesetzt werden, um den Ablauf zu automatisieren.

- Zur Ressourcen-Einsatzplanung
  Ablaufmodelle können dazu verwendet werden, Entscheidungen über den Einsatz von personellen und maschinellen Aufgabenträgern zu treffen.
- Zur Unterstützung der Ablaufausführung durch ein Workflow-Managementsystem
  Eine automatisierte Ablaufausführung setzt eine präzise formale Beschreibung voraus, welche durch eine Modellierung des Ablaufs erreicht werden kann.
- Zur Überwachung und Steuerung von Abläufen
  Ein Ablaufschema kann dazu verwendet werden, eine konkrete Ablaufausprägung durch einen Soll-/Ist-Vergleich auf unerwünschte Abweichungen zu untersuchen. Dieser Zweck der Modellierung wird in dieser Arbeit verfolgt.

Nachdem der Zweck der Modellierung vorgestellt wurde, wird im nachfolgenden Kapitel die Modellierung von Abläufen beschrieben.

4.2 Modellierung von Abläufen

Zur Modellierung von Abläufen existieren eine Vielzahl von Modellierungsmethoden, z.B. Ereignisgesteuerte Prozessketten (EPKs) [KeNS92], die Business Process Model and Notation (BPMN) [Mode11] oder Petri-Netz-basierte Sprachen [ElNu93]. Eine vollständige Auflistung und Beschreibung aller existierenden Modellierungssprachen würde den Rahmen dieser Arbeit sprengen. Deshalb wird in diesem Kapitel der Fokus auf die Modellierung von Abläufen mit Petri-Netzen gelegt, mit denen eine formale, graphische Beschreibung von Abläufen möglich ist. Aufgrund dieser Eigenschaften von Petri-Netzen ist es nicht nur möglich, Abläufe zu modellieren, sondern Petri-Netze auch hinsichtlich verschiedener Aspekte zu analysieren und in andere Modelle zu transformieren.

4.2.1 Petri-Netze

Petri-Netze sind ein Formalismus zur Modellierung von Abläufen und basieren auf der Dissertation „Kommunikation mit Automaten“ von Carl Adam Petri [Petr62]. Seit dieser Dissertation sind verschiedene Weiterentwicklungen von Petri-Netzen zu höheren Petri-Netzen, als auch Modellierungsmethoden auf Basis von Petri-Netzen entwickelt worden. Petri-Netze werden in verschiedenen Forschungsbereichen, wie beispielsweise Datenbanksysteme, Software Engineering oder Geschäftsprozessmodellierung angewendet [DeOb96].

Petri-Netze ermöglichen die Beschreibung von sequenziellen, nebenläufigen, aber auch von sich gegenseitig ausschließenden Aktivitäten [Ober96, S.98]. Zur Beschreibung werden

Kreise, Vierecke und Pfeile verwendet. Kreise werden als Stellen bezeichnet und repräsentieren statische Objekte, z.B. Dokumente, Daten, etc. Vierecke werden als Transitionen bezeichnet und repräsentieren Aktivitäten, Ereignisse oder lokale Zustandsübergänge [Ober96, S.98]. Stellen und Transitionen bilden zusammen die Knoten des Petri-Netzes. Die Pfeile repräsentieren Flußrelationen, werden als Kanten bezeichnet, und verbinden Stellen und Transitionen miteinander [Baum97, S.50]. Ein Petri-Netz ist ein bipartiter Graph [Baum97, S.50], d.h. dass seine Knoten in zwei disjunkte Teilmengen (Stellen und Transitionen) aufgeteilt werden können, und innerhalb der Knoten dieser Teilmengen keine Kanten verlaufen. Formal ist ein Petri-Netz wie folgt definiert (die Definitionen und nachfolgenden Beschreibungen in diesem Kapitel sind aus [Baum97, Ober96] übernommen):

Definition 4.1: Netz

Ein Netz ist ein Tripel $N = (S, T, F)$, für das gilt:

(i) $S \cap T = \emptyset$
(ii) $S \cup T \neq \emptyset$
(iii) $F \subseteq (S \times T) \cup (T \times S)$

Die Elemente der Menge S heißen *Stellen*. Die Elemente der Menge T heißen *Transitionen*. Die Elemente der Flußrelation F heißen *Kanten*.

■

Abbildung 4.1 zeigt ein beispielhaftes Netz $N = (S, T, F)$ mit $S = \{s1, s2, s3, s4, s5\}$, $T = \{t1, t2\}$ und $F = \{(s1, t1), (t1, s2), (t1, s3), (s3, t2), (s2, t2), (s4, t2), (t2, s5)\}$.

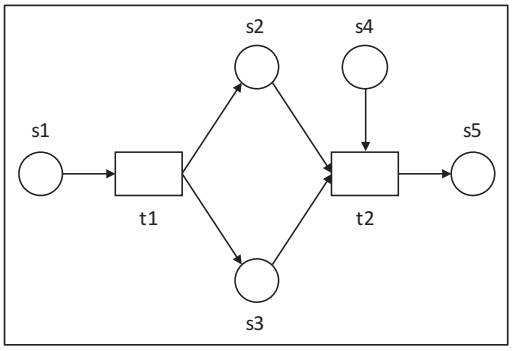


Abbildung 4.1: Beispielhaftes Netz

Zur Beschreibung von Nachbarknoten in einem Netz werden die Begriffe Vorbereich und Nachbereich verwendet. Der Vorbereich beschreibt alle Knoten, die durch eine ausgehende

Kante mit dem Knoten x verbunden sind. Der Nachbereich beschreibt alle Knoten, die über eine eingehende Kante mit dem Knoten x verbunden sind. Vor- und Nachbereich sind wie folgt definiert:

Definition 4.2: Vorbereich, Nachbereich

Gegeben sei ein Netz $N = (S, T, F)$. Es sei $X = S \cup T$.

(i) $\cdot x = \{y \in X \mid (y, x) \in F\}$ ist der Vorbereich von x
(ii) $x \cdot = \{y \in X \mid (x, y) \in F\}$ ist der Nachbereich von x

■

In Abbildung 4.1 ist der Vorbereich von t1 $\cdot t1 = \{s1\}$ und der Nachbereich von t1 $t1 \cdot = \{s2, s3\}$.

Bei der Modellierung von Petri-Netzen kann schrittweise vorgegangen werden. Dabei spielen Netz-Transformationen eine wichtige Rolle, die unterschieden werden in Vergröberung/Verfeinerung, Einbettung/Restriktion und Faltung/Entfaltung [Ober96, S.287]. Ein Teil-Netz ist wie folgt definiert:

Definition 4.3: Teil-Netz

Ein Netz $N' = (S', T', F')$ ist ein *Teil-Netz* des Netzes $N = (S, T, F)$, wenn gilt:

(i) $S' \subseteq S$
(ii) $T' \subseteq T$
(iii) $F' = F \cap ((S' \times T') \cup (T' \times S'))$

Der *Rand* eines Teil-Netzes in Bezug zum Gesamtnetz sind diejenigen seiner Knoten, die mit Kanten zum Restnetz verbunden sind:

$Rand(N', N) = \{x \in S' \cup T' | (x \cdot \cup \cdot x) \setminus (S' \cup T') \neq \emptyset\}$

(Der Nachbereich $x \cdot$ bzw. Vorbereich $\cdot x$ bezieht sich in dieser Definition auf das Gesamtnetz N).

Ein Teil-Netz heißt *transitionsberandet*, wenn $Rand(N', N) \subseteq T'$ und *stellenberandet*, wenn $Rand(N', N) \subseteq S'$.

■

In Abbildung 4.2 ist ein stellenberandetes Teil-Netz N' dargestellt.

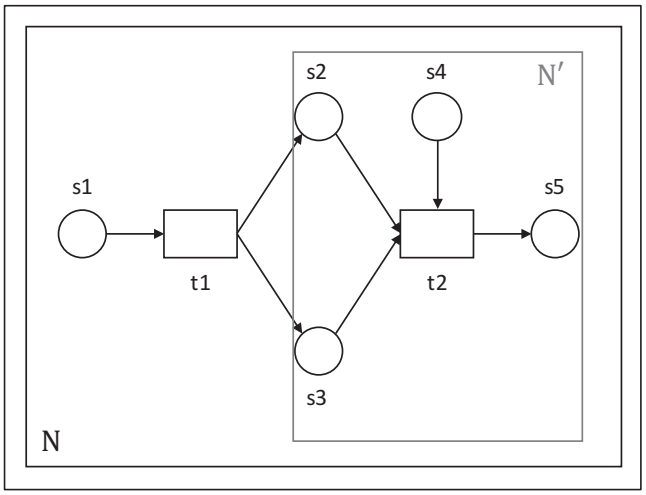


Abbildung 4.2: Stellenberandetes Teil-Netz N’

VERGRÖBERUNG/VERFEINERUNG

Bei einer Vergröberung wird ein transitionsberandetes bzw. stellenberandetes Teil-Netz durch eine einzelne Transition bzw. Stelle ersetzt. Die Verfeinerung ist die Umkehrung der Vergröberung und bezeichnet das Ersetzen einer Transition bzw. Stelle durch ein transitionsberandetes bzw. stellenberandetes Teil-Netz. Die Vergröberung bzw. Verfeinerung wird eingesetzt, um von bestimmten Zusammenhängen in einem Netz zu abstrahieren, bzw. diese zu detaillieren. Abbildung 4.3 zeigt eine Vergröberung des stellenberandeten Teil-Netzes N' in die einzelne Stelle $s5$ und die Umkehrung davon.

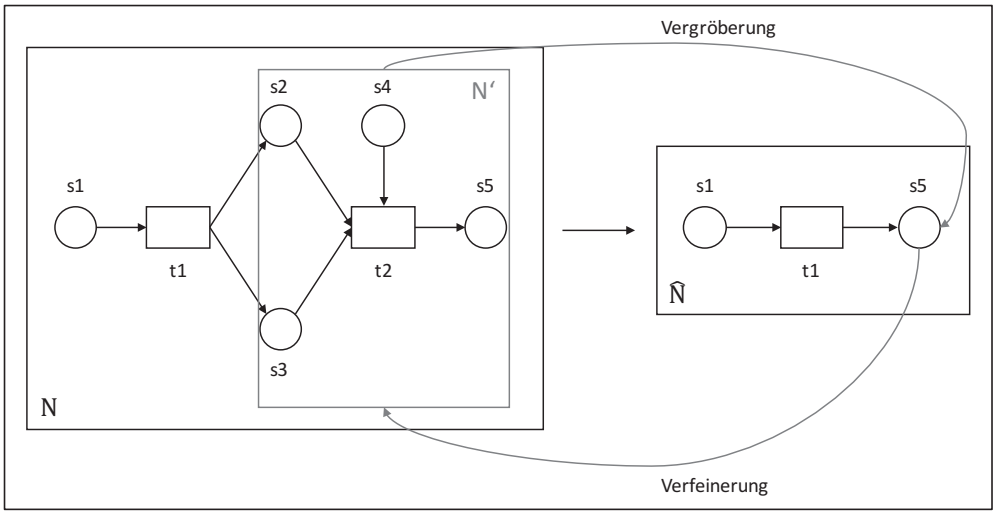


Abbildung 4.3: Vergröberung/Verfeinerung eines Netzes

EINBETTUNG/RESTRIKTION

Die Einbettung eines Netzes bezeichnet die Erweiterung eines Netzes um weitere Knoten und Kanten, so dass das ursprüngliche Netz ein Teil-Netz des neu entstandenen Netzes ist. Die Restriktion stellt die Umkehrung einer Einbettung dar und wird zur Restriktion eines Netzes in ein Teil-Netz verwendet. Die Einbettung wird eingesetzt, um zusätzliche Aspekte in einem Modell zu berücksichtigen, beispielsweise Umweltaspekte oder andere Umgebungsaspekte. Abbildung 4.4 zeigt die Einbettung des Netzes N in das Netz $\widehat{N}$ bzw. die Restriktion von $\widehat{N}$ zu N.

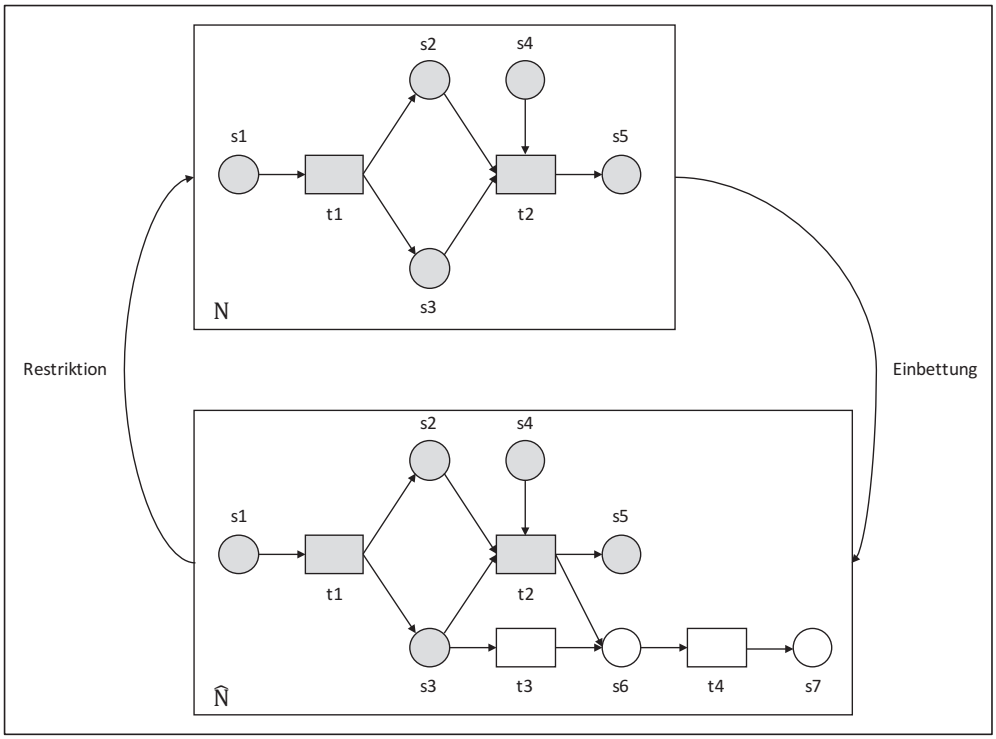


Abbildung 4.4: Einbettung/Restriktion eines Netzes

FALTUNG/ENTFALTUNG

Mit der Faltung eines Netzes wird das „Aufeinanderfalten“ gleichartiger Teil-Netze eines Netzes verstanden. Dabei werden Knoten gleichen Typs gefaltet, so dass die Flußrelation gewahrt bleibt. Die Umkehrung der Faltung wird als Entfaltung bezeichnet. Faltungen werden für globale Abstraktionen durchgeführt, beispielsweise zur Abstraktion aller Jahresanfänge wie „1.1.2000“, „1.1.2001“, ... zu einer einzelnen Stelle „1.1“. Abbildung 4.5 zeigt die Faltung des Netzes N in das Netz N'.

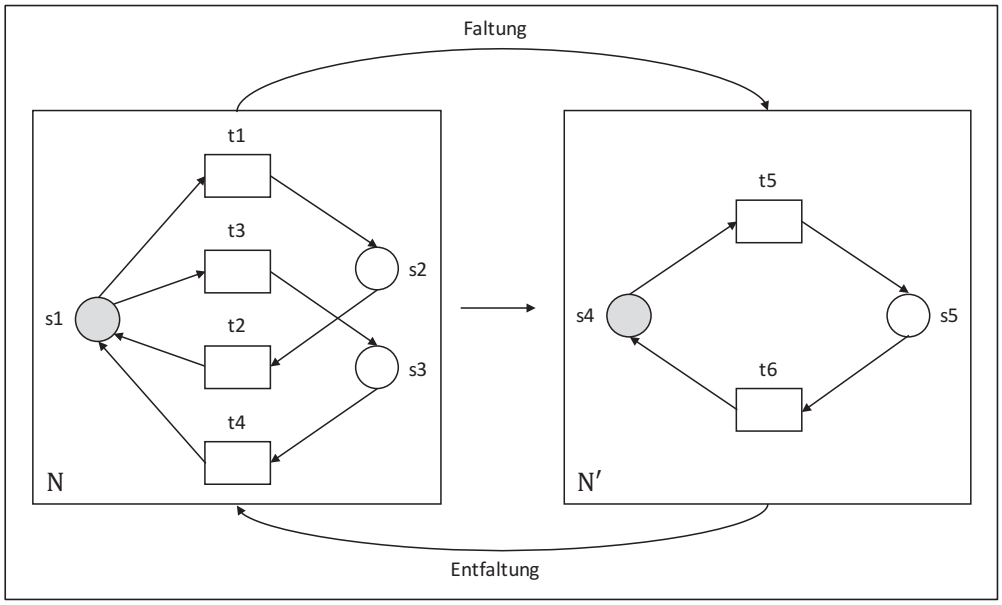


Abbildung 4.5: Faltung/Entfaltung eines Netzes

Nachdem Netze in diesem Unterkapitel definiert und verschiedene Transformationen von Netzen beschrieben wurden, wird nachfolgend eine Auswahl einfacher und höherer Varianten von Netzen vorgestellt, mit denen zusätzlich Zustände und Zustandsübergänge beschrieben werden können. Einfache Petri-Netze zeichnen sich dadurch aus, dass die verwendeten Marken anonym und somit nicht unterscheidbar sind. Höhere Petri-Netze ermöglichen die Beschreibung individueller und somit unterscheidbarer Marken [Reis13, S.18]. Marken werden als ausgefüllte Kreise dargestellt und können als Objekte interpretiert werden, die auf den Stellen eines Netzes liegen und durch Transitionen verbraucht werden [Baum97, S.77f.].

4.2.2 Einfache Petri-Netze

Als Varianten von Netzen, die als einfache Petri-Netze bezeichnet werden, werden nachfolgend Bedingungs/Ereignis-Netze und Stellen/Transitionen-Netze beschrieben. Bedingungs/Ereignis-Netze erweitern Netze um die Möglichkeit, dass Stellen markiert oder unmarkiert sein können. Stellen/Transitionen-Netze bieten zusätzlich die Möglichkeit mehr als nur eine Marke auf einer Stelle abzulegen [Reis13, S.18]. Bedingungs/Ereignis-Netze sind wie folgt definiert (die nachfolgenden Definitionen sind aus [Ober96] übernommen):

Definition 4.4: Bedingungs/Ereignis-Netz

Ein Bedingungs/Ereignis-Netz ist ein Tupel $BEN = (S, T, F, M^0)$, für das gilt:

(i) (S, T, F) ist ein Netz.
(ii) $M: S \rightarrow \{0,1\}$ weist den Stellen im Netz eine Markierung zu. Die Start-Markierung ist M^0.

■

Wenn eine Stelle s markiert ist, d.h. $M^s = 1$, dann ist die entsprechende Bedingung $wahr$. Wenn alle Stellen im Vorbereich einer Transition t markiert sind und alle Stellen im Nachbereich von t nicht markiert sind, dann ist t aktiviert und kann schalten bzw. ausgeführt werden. Beim Schalten von t werden aus den Stellen des Vorbereichs von t alle Marken entnommen, und in allen Stellen des Nachbereichs von t jeweils eine Marke abgelegt. Solche Schaltungen ändern den Wahrheitswert von Bedingungen.

Im Gegensatz zu Bedingungs/Ereignis-Netzen bieten Stellen/Transitionen-Netze die Möglichkeit, mehr als nur eine Marke auf einer Stelle abzulegen und zusätzlich Kanten mit sogenannten Kantengewichten zu erweitern. Es sind sowohl Stellen mit begrenzter, als auch unbegrenzter Kapazität erlaubt. Transitionen in einem Stellen/Transitionen-Netz können schalten, wenn im Vorbereich der Transitionen mindestens die Anzahl Marken je Stelle gemäß den eingehenden Kantengewichten der Transition vorliegen, und im Nachbereich alle Marken entsprechend den ausgehenden Kantengewichten der Transition und den Kapazitäten in den jeweiligen Stellen abgelegt werden können. Beim Schalten werden die Marken entsprechend den eingehenden Kantengewichten verbraucht und neue Marken entsprechend den ausgehenden Kantengewichten erzeugt [Baum97, S.77f.]. Formal sind Stellen/Transitionen-Netze wie folgt definiert:

Definition 4.5: Stellen/Transitionen-Netz

Ein Stellen/Transitionen-Netz ist ein Tupel $ST = (S, T, F, K, W, M^0)$, für das gilt:

(i) (S, T, F) ist ein Netz,
(ii) $K: S \rightarrow \mathbb{N} \cup \{\infty\}$ weist jeder Stelle eine Aufnahmekapazität für Marken zu, welche nicht überschritten werden darf,
(iii) $W: F \rightarrow \mathbb{N}$ weist jeder Kante im Netz eine Gewichtung zu,
(iv) $M: S \rightarrow \mathbb{N}_0$, wobei $\forall s \in S: M(s) \leq K(s)$, weist jeder Stelle eine bestimmte Anzahl von Marken zu. M^0 ist die Startmarkierung.

■

Die Schaltregel für Stellen/Transitionen-Netze ist wie folgt definiert:

Definition 4.6: Schaltregel für Stellen/Transitionen-Netze

Gegeben sei ein Stellen/Transitionen-Netz $ST = (S, T, F, K, W, M^0)$.

Eine Transition $t \in T$ ist aktiviert unter einer Markierung M, wenn gilt:

(i) $\forall s \in \cdot\, t\colon M(s) \geq W(s,t)$,
(ii) $\forall s \in t \cdot\colon M(s) \leq K(s) - W(t,s)$.

Eine Transition t kann schalten, wenn sie unter M aktiviert ist. Das Schalten von t unter M überführt M in die Folgemarkierung M'. Dabei gilt:

$$M'(s) = \begin{cases} M(s) - W(s,t) & \text{falls } s \in \cdot\, t \text{ und } s \notin t \cdot \\ M(s) + W(t,s) & \text{falls } s \in t \cdot \text{ und } s \notin \cdot\, t \\ M(s) - W(s,t) + W(t,s) & \text{falls } s \in t \cdot \text{ und } s \in \cdot\, t \\ M(s) & \text{sonst} \end{cases}$$

■

Abbildung 4.6 zeigt eine Markierung M und die Folgemarkierung M' nach dem Schalten von $t1$.

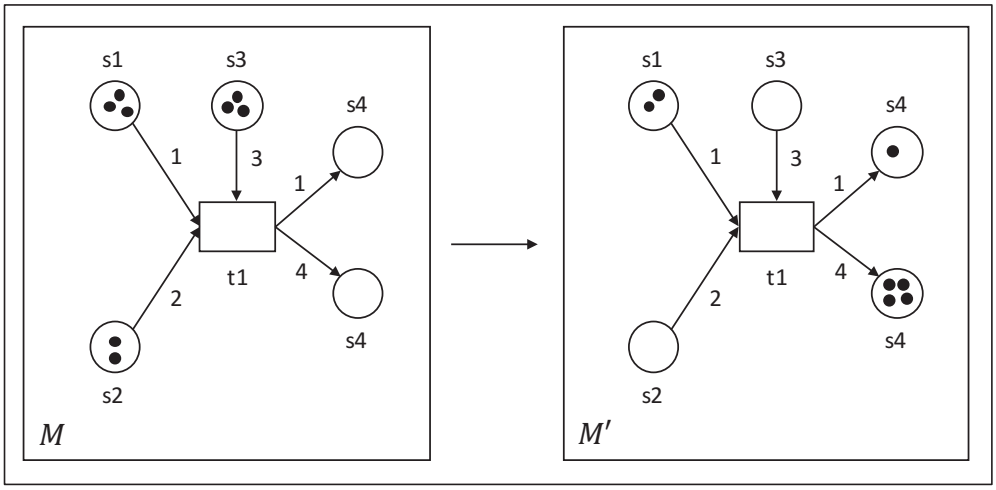


Abbildung 4.6: Markierung und Folgemarkierung eines Netzes

In Petri-Netzen lassen sich verschiedene elementare Ablaufmuster modellieren, die nachfolgend beschrieben werden. Für eine Übersicht über komplexere Ablaufmuster siehe [ATRD15].

Sequenz

Eine Sequenz beschreibt ein Ablaufmuster, wenn genau eine Transition vor einer anderen Transition geschaltet werden muss.

XOR-Split

Ein XOR-Split beschreibt einen Konflikt zwischen Transitionen, die sich dieselbe Stelle im Vorbereich teilen. Um eine Marke zu verbrauchen, kann nur eine der Transitionen schalten.

XOR-Join

Der XOR-Join ist das logische Gegenstück zum XOR-Split. Beim XOR-Join haben Transitionen dieselbe Stelle als Nachbereich. Dieses Ablaufmuster wird auch als Kontakt bezeichnet.

AND-Split

Mit einem AND-Split kann eine Nebenläufigkeit von Transitionen modelliert werden. Dazu muss eine Transition mindestens zwei Stellen im Nachbereich haben, die wiederum als Vorbereich von jeweils mindestens einer Transition dienen.

AND-Join

Beim AND-Join findet eine Synchronisation von nebenläufigen Transitionen statt. Mindestens zwei Transitionen haben jeweils eine Stelle im Nachbereich, die jeweils Eingangsstellen einer weiteren Transition sind. Abbildung 4.7 veranschaulicht die fünf elementaren Ablaufmuster in Petri-Netzen.

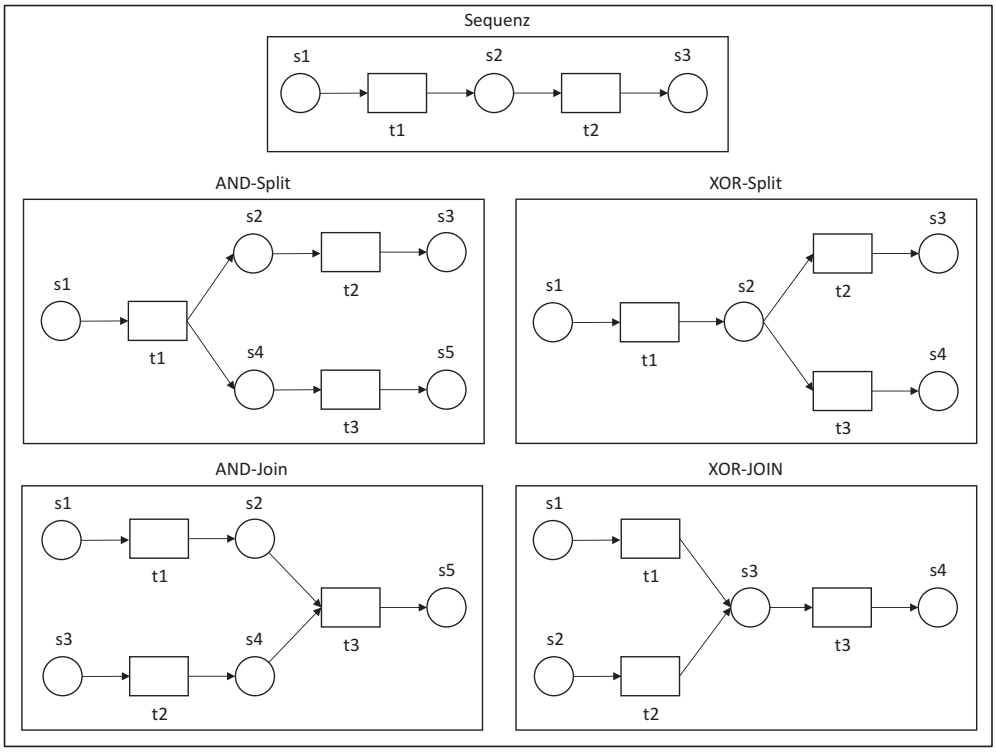


Abbildung 4.7: Elementare Ablaufmuster

4.2.3 Höhere Petri-Netze

Nachdem im Kapitel zuvor einfache Petri-Netze vorgestellt wurden, werden nachfolgend höhere Petri-Netze beschrieben. Im Gegensatz zu einfachen Petri-Netzen eignen sich höhere Petri-Netze aufgrund der Modellierbarkeit von objektbezogenen Aspekten besonders für die Modellierung betrieblicher Abläufe [Ober96, S.98ff.]. Zudem ist mit höheren Petri-Netzen eine kompaktere Darstellung von Sachverhalten möglich, als mit einfachen Petri-Netzen [Baum97, S.193]. Varianten von höheren Petri-Netzen sind Prädikate/Transitionen-Netze und XML-Netze.

In Prädikaten/Transitionen-Netzen [GeLa81, Genr86] stellen Kreise Prädikate (Relationsschemata) und Vierecke Transitionen dar. Prädikate repräsentieren Relationsschemata und Marken repräsentieren konkrete Relationen des entsprechenden Typs [Ober96, S.103]. Relationen stellen Wertetupel dar, und können sich in ihrer Struktur und den Attributwerten unterscheiden [Lenz03, S.14]. Transitionen entsprechen Klassen von Operationen auf

den Ausgangs- und Eingangs-Prädikaten [Ober96, S.103]. Kanten werden Kantenbeschriftungen zugeordnet, die den Kanten Mengen von Variablentupeln zuweisen [Ober96, S.293]. Transaktionen sind aktiviert, wenn in den eingehenden Prädikaten die gemäß der entsprechenden Kantenbeschriftung benötigten Tupel vorhanden sind und in den ausgehenden Prädikaten, die gemäß der jeweiligen Kantenbeschriftung definierten Tupel nicht vorhanden sind [Ober96, S.103]. Zusätzlich zu den Kantenbeschriftungen können Transitionsbeschriftungen angegeben werden, mit denen weitere Bedingungen für die betroffenen Tupel angegeben werden können [Ober96, S.103]. Wenn eine Transition schaltet, werden die aufgrund der Kanten- und Transitionsbeschriftung ausgewählten Tupel in den Eingangs-Prädikaten gelöscht und in den Ausgangs-Prädikaten hinzugefügt [Ober96, S.103]. Formal sind Prädikaten/Transitionen-Netze wie folgt definiert (die nachfolgenden Definitionen sind aus [Ober96] übernommen):

Definition 4.7: Prädikate/Transitionen-Netz

Ein Prädikate/Transitionen-Netz ist ein Tupel $PT = (S, T, F, \Psi, KB, TI, M^0)$, für das gilt:

(i) (S, T, F) ist ein Netz. S wird als Menge von Prädikaten (Relationsschemata) mit potenziell unterschiedlichen Ausprägungen (Relationen) und T als Menge von Transitionsschemata interpretiert.

(ii) $\Psi = (D, FU, PR)$ ist eine Struktur, die aus endlichen Individuenmengen D, einer Menge FU von auf D definierten Funktionen, sowie einer Menge PR von auf D definierten Prädikaten mit unveränderlichen Ausprägungen besteht.

(iii) KB beschriftet Kanten aus F mit Mengen von Variablentupeln, deren Stelligkeit der Stelligkeit des adjazenten Prädikats entspricht.

(iv) TI weist jeder Transition $t \in T$ einen in Ψ gebildeten prädikatenlogischen Ausdruck als Inschrift zu. Jede Variable in diesem Ausdruck muss in der Beschriftung einer zur Transition adjazenten Kante vorkommen.

(v) M ist eine Markierung der Prädikate mit konstanten Individuentupeln, die der Stelligkeit des entsprechenden Prädikats entsprechen. M^0 ist die Startmarkierung.

■

Definition 4.8: Prädikatenlogischer Ausdruck

P_ψ ist die kleinste Menge prädikatenlogischer Ausdrücke zu einer Struktur $\Psi = (D, FU, PR)$, für die gilt:

(i) $TRUE$ ist ein prädikatenlogischer Ausdruck.
(ii) P sei ein n-stelliges Prädikat in PR und $t_1, \dots, t_n$ seien Terme. Dann ist $P(t_1, \dots, t_n)$ ein prädikatenlogischer Ausdruck.
(iii) Wenn p_1, p_2 prädikatenlogische Ausdrücke sind, dann sind auch $p_1 \wedge p_2$, $p_1 \vee p_2$, $p_1 \rightarrow p_2$ und $\neg p_1$ prädikatenlogische Ausdrücke.

■

Abbildung 4.8 zeigt ein beispielhaftes Prädikate/Transitionen-Netz zur Zuweisung eines RFCs zu einem Mitarbeiter, um diesen zu prüfen. Es sind die Prädikate RFC, MITARBEITER und IN PRÜFUNG abgebildet, die mit den entsprechenden Individuentupeln markiert sind. Die Transition RFC-ZUWEISUNG entfernt jeweils ein Tupel aus dem Prädikat RFC (RFC-ID, IT-SERVICE, BESCHREIBUNG) und MITARBEITER (MITARBEITER, IT-SERVICE, KAPAZITÄT), wenn der entsprechende Mitarbeiter noch ausreichend KAPAZITÄT hat, und erstellt ein Tupel im Prädikat IN PRÜFUNG (RFC-ID, IT-SERVICE, BESCHREIBUNG, MITARBEITER), das den zugeordneten MITARBEITER zum jeweiligen RFC enthält, damit dieser den RFC prüfen kann. Zusätzlich wird ein neues Tupel im Prädikat MITARBEITER (MITARBEITER, IT-SERVICE, KAPAZITÄT') erstellt, das eine um eine Einheit verringerte KAPAZITÄT des zugewiesenen MITARBEITERS enthält.

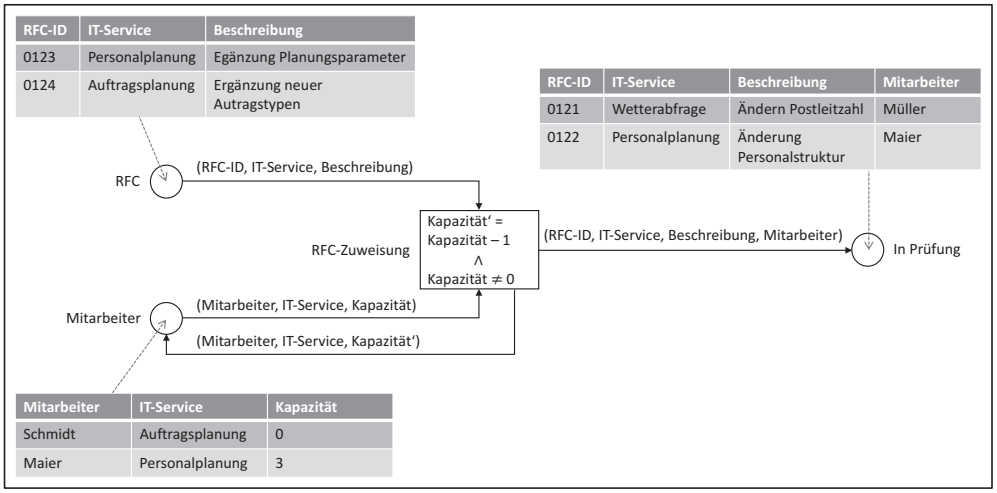


Abbildung 4.8: Beispielhaftes Prädikate/Transitionen-Netz

Eine weitere Variante höherer Petri-Netze sind XML-Netze, die auf dem Austausch von XML-Dokumenten basieren, die Geschäftsobjekte repräsentieren [Lenz03, LeOb03]. Wie in Kapitel 4.3.2 beschrieben wird, können IT-Services bzw. der Aufbau eines IT-Services ebenfalls in Form von XML-Dokumenten beschrieben werden, wodurch sich XML-Netze potenziell sehr gut für die Modellierung von IT-Service-Wartungen eignen.

Stellen repräsentieren in einem XML-Netz Behälter für XML-Dokumente, die durch Stellenschemata typisiert sind, d.h. Stellen können nur solche XML-Dokumente enthalten, die hinsichtlich des entsprechenden XML-Schemata der Stelle gültig sind. Marken stellen in XML-Netzen tatsächliche XML-Dokumente dar. Die Menge der XML-Dokumente in einer Stelle stellt die Markierung der Stelle dar. Aufgrund der Verwendung von XML-Schemata können auch komplex strukturierte Objekte modelliert werden [Lenz03, S.172].

Transitionen repräsentieren in XML-Netzen aktive Komponenten, mit denen markierte XML-Dokumente aus den Stellen des Vorbereichs einer Transition gelesen, oder teilweise bzw. ganz gelöscht werden können. Für Stellen aus dem Nachbereich einer Transition können neue XML-Dokumente erzeugt oder Teile in XML-Dokumente der Markierung eingesetzt werden. Transitionen können zusätzlich Transitionsinschriften enthalten, um die möglichen Operationen einzuschränken. Transitionsinschriften sind wie in Prädikate/Transitionen-Netzen logische Ausdrücke, die erfüllt sein müssen, damit eine Transition schalten kann [Lenz03, S.172f.].

Den Kanten werden als Kanteninschriften sogenannte Filterschemata zugeordnet, die mit den XML-Schemata der adjazenten Stellen übereinstimmen müssen und die Selektion von Teilmengen, der markierten XML-Dokumente, präzisieren. Im Gegensatz zu den Stellenschemata ermöglichen Filterschemata nicht nur die Beschreibung des Aufbaus eines XML-Dokuments bzw. seiner Struktur, sondern auch das Hinzufügen, Löschen oder Manipulieren seiner Inhalte bzw. ganzer XML-Dokumente [Lenz03, S.173].

Übernommen aus [Lenz03] wird ein XML-Netz wie folgt formal definiert:

Definition 4.9: XML-Netz

Ein XML-Netz ist ein Tupel $XN = (S, T, F, \Psi, SI, KB, TI, M^0)$, für das gilt:

(i) (S, T, F) ist ein Petri-Netz.

(ii) $\Psi = (D, FU, PR)$ ist eine Struktur, die aus endlichen Individuenmengen D, einer Menge FU von auf D definierten Funktionen, sowie einer Menge PR von auf D definierten Prädikaten mit unveränderlichen Ausprägungen besteht.

(iii) Die Funktion $SI: S \rightarrow \mathbb{G}_{el}$ weist jeder Stelle $s \in S$ ein elementares XML-Schema $G \in \mathbb{G}_{el}$ als Stellentypisierung zu.

(iv) KB beschriftet Kanten aus F mit einer Menge valider, bezüglich des adjazenten XML-Schemas zulässiger Filterschemata.

(v) TI weist jeder Transition $t \in T$ einen in Ψ gebildeten prädikatenlogischen Ausdruck als Inschrift zu. Jede Variable in diesem Ausdruck muss in der Beschriftung einer zur Transition adjazenten Kante vorkommen.

(vi) M markiert die Stellen $s \in S$ mit, bezüglich des XML-Schemas gültigen, XML-Dokumenten. M^0 ist die Startmarkierung.

■

In der Definition von XML-Netzen von Lenz [Lenz03] wird für die Beschreibung von Filterschemata die Sprache XML Manipulation Language (XManiLa) verwendet, die an die Notation von XML-Schema angelehnt ist [Lenz03, S.133]. XManiLa ermöglicht die graphische Darstellung von Selektionen und Manipulationen von XML-Dokumenten, und nutzt dafür Element- sowie Attributfilter, die den Element- bzw. Attributtypen in XML-Schema entsprechen. Elementfilter können, analog der Nutzung von Attributen in Elementen bei XML-Schema, Attributfilter enthalten. Element- und Attributfilter können Konstanten oder Variablen enthalten, um bestimme XML-Dokumente auswählen zu können. Elementfilter werden als Rechtecke dargestellt. Attributfilter werden als Rechtecke unter dem Elementfilter dargestellt. Elementfilter können zudem unterschieden werden in Vergleichsfilter, die lediglich zur Selektion von Instanzen eines bestimmten Elementtyps verwendet werden, und Manipulationsfilter, die zur Manipulation von XML-Dokumenten verwendet werden. Manipulationsfilter werden graphisch durch einen schwarzen Balken auf der linken Seite des Rechtecks dargestellt. Wenn ein XML-Dokument nur gelesen werden soll, werden Vergleichsfilter verwendet. Bei der Manipulation von XML-Dokumenten werden Manipulationsfilter verwendet. Dabei bedeutet ein Manipulationsfilter, der eine aus einer Transition ausgehende Kante beschriftet, dass es sich um eine Einfügeoperation handelt. Ein Manipulationsfilter, der eine aus einer Stelle ausgehende Kante beschriftet, beschreibt eine Löschoperation [LeOb02]. Abbildung 4.9 veranschaulicht die drei XManiLa-Anfragen Lesen, Einfügen und Löschen mit Elementfiltern.

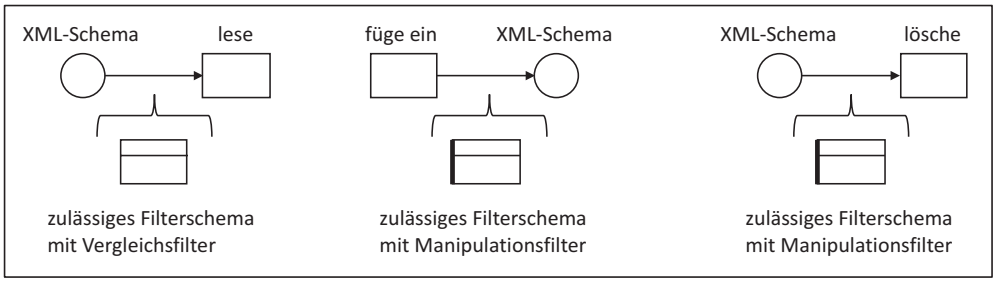


Abbildung 4.9: XManiLa-Anfragen (angelehnt an [LeOb02])

Abbildung 4.10 zeigt ein beispielhaftes XML-Netz, in dem die Reservierung eines Hotelzimmers modelliert ist und die Filterschemata als XManiLa-Anfragen beschrieben sind.

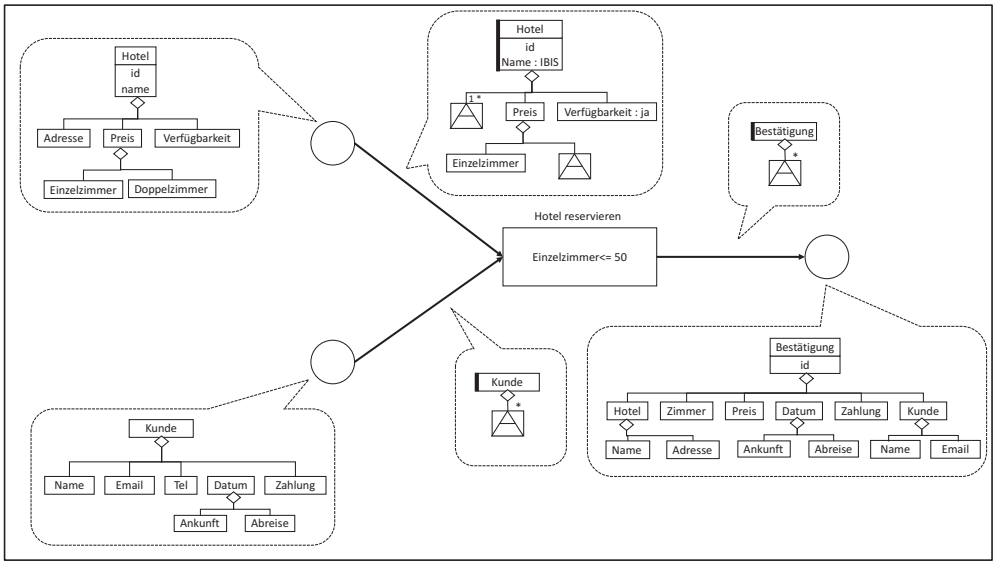
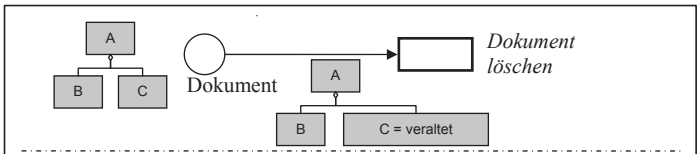


Abbildung 4.10: Beispielhaftes XML-Netz (angelehnt an [CLOS09])

Statt XManiLa kann auch die Sprache XML Query (XQuery) zur Beschreibung von Filterschemata eingesetzt werden, wie in [Mevi06] beschrieben. XQuery ist eine Anfragesprache für XML-Dokumente, wird vom World Wide Web Consortium (W3C) entwickelt und liegt aktuell in Version 3.0 vor [Wc14]. Abbildung 4.11 zeigt eine exemplarische Löschoperation für ein Dokument, das aus den Knoten A, B und C besteht. Wenn der Knoten C den Wert „veraltet“ enthält, soll dieses Dokument gelöscht werden. Der entsprechende XQuery-Ausdruck zur Selektion des Dokuments ist im unteren Teil des Dokuments abgebildet.




```
let $dummyvar:=""
for $Dokument1_A in
doc("file:/DokumentLoeschen/xml/Dokument1.xml")/A
let $Dokument1_AB:= $Dokument1_A/*[1]/self::B
let $Dokument1_AC:= $Dokument1_A/*[2]/self::C

where (1=1
     and exists($Dokument1_A)
     and exists($Dokument1_AB)
     and exists($Dokument1_AC)
     and $Dokument1_AC="veraltet")
return (<success/>)
```

Abbildung 4.11: Beispielhafte XQuery-Anfrage [Mevi06]

Nachdem die Modellierung von Abläufen beschrieben wurde, werden nachfolgend Methoden zur Modellierung der zu wartenden Anwendungssoftware, die zur Ausführung der Aktivitäten des IT-Service-Anbieters benötigt werden, beschrieben.

4.3 Modellierung von IT-Service-Komponenten

Nachfolgend werden Methoden zur Modellierung von IT-Service-Komponenten beschrieben. In ITIL ist die Modellierung und Pflege von IT-Service-Komponenten Gegenstand des Service Asset and Configuration Managements (siehe Kapitel 2.5.3). Allerdings legt ITIL keine zu nutzende Modellierungssprache für IT-Service-Komponenten fest. Zur Modellierung von IT-Service-Komponenten wurden verschiedene Modellierungssprachen entwickelt, mit denen eine eher präskriptive oder deskriptive Modellierungsperspektive verfolgt wird (vgl. [Lude02]). Präskriptive Modelle spezifizieren die Struktur und die technischen Details zu erstellender IT-Services bzw. IT-Service-Komponenten, wohingegen deskriptive Modelle zur Beschreibung des Zustands, sich bereits in Betrieb befindender IT-Services, verwendet werden. Jedoch können häufig auch Modellarten mit einem präskriptiven Fokus zur Beschreibung eines IT-Services aus deskriptiver Perspektive verwendet werden [Brei16, S.27f.]. Für die Dokumentation von detaillierten Laufzeitinformationen von IT-Service-Komponenten eignen sich insbesondere IT-Instanzmodelle, die den genauen, aktuellen Status, der sich in Betrieb befindenden IT-Service-Komponenten, repräsentieren [Binz15, 45f.]. Der Detaillierungsgrad präskriptiver und deskriptiver Modellierungssprachen zur Beschreibung von IT-Service-Komponenten ist generell jedoch sehr ähnlich [Binz15, S.46]. Präskriptive

Modellierungssprachen sind beispielsweise TOSCA [Oasi13a] oder HEAT [Open16]. Beispiele für deskriptive Beschreibungssprachen sind das Common Information Model (CIM) [Dist16] oder die IT Infrastructure Modelling Language (ITML) [FHKF09]. Nachfolgend werden die Topology and Orchestration Specification for Cloud Applications (TOSCA) und CIM, aufgrund ihres hohen Standardisierungsgrads und ihrer Verbreitung im Vergleich zu ähnlichen Sprachen, genauer vorgestellt[1].

4.3.1 Common Information Model (CIM)

Das Common Information Model ist ein von der Distributed Management Task Force, Inc. (DMTF) [Dist16] entwickelter Standard zur Modellierung technischer Systemumgebungen, wie beispielsweise dem Netzwerk, Anwendungen, aber auch Hardwareresourcen. Es verwendet dabei einen objektorientierten Ansatz, nutzt insbesondere Vererbungen und wird mit der Unified Modeling Language (UML) beschrieben. In CIM werden IT-Service-Komponenten und ihre Beziehungen mit einem CIM Management Schema modelliert. Elemente (*Element*) repräsentieren eigenständige IT-Service-Komponenten, z.B. eine Applikation oder eine Hardwarekomponente, und beinhalten Beschreibungen über die Konfiguration des Elements (*Property*), als auch seines Lebensstatus (*Method*). Abbildung 4.12 stellt das Metamodel und den Aufbau eines Elements dar. CIM ist in ein Core Model, Common Model und Extension Schemas aufgeteilt. Das Core Model beinhaltet eine kleine Menge vordefinierter abstrakter Klassen und ihre Eigenschaften und Beziehungen untereinander, die als Basis genutzt werden können, um IT-Service-Komponenten zu beschreiben. Das Common Model bietet eine Erweiterung des Core Models um vordefinierte technologieunabhängige Klassen wie Anwendungen, Anwendungsserver, Datenbanken, Netzwerke und weitere. Extension Schemas erweitern das Common Model um technologie-spezifische Klassen, beispielsweise einen Windows oder Unix Server [Dist12a].

[1] Für eine Übersicht über vergleichbare Sprachen siehe [BrSW14]

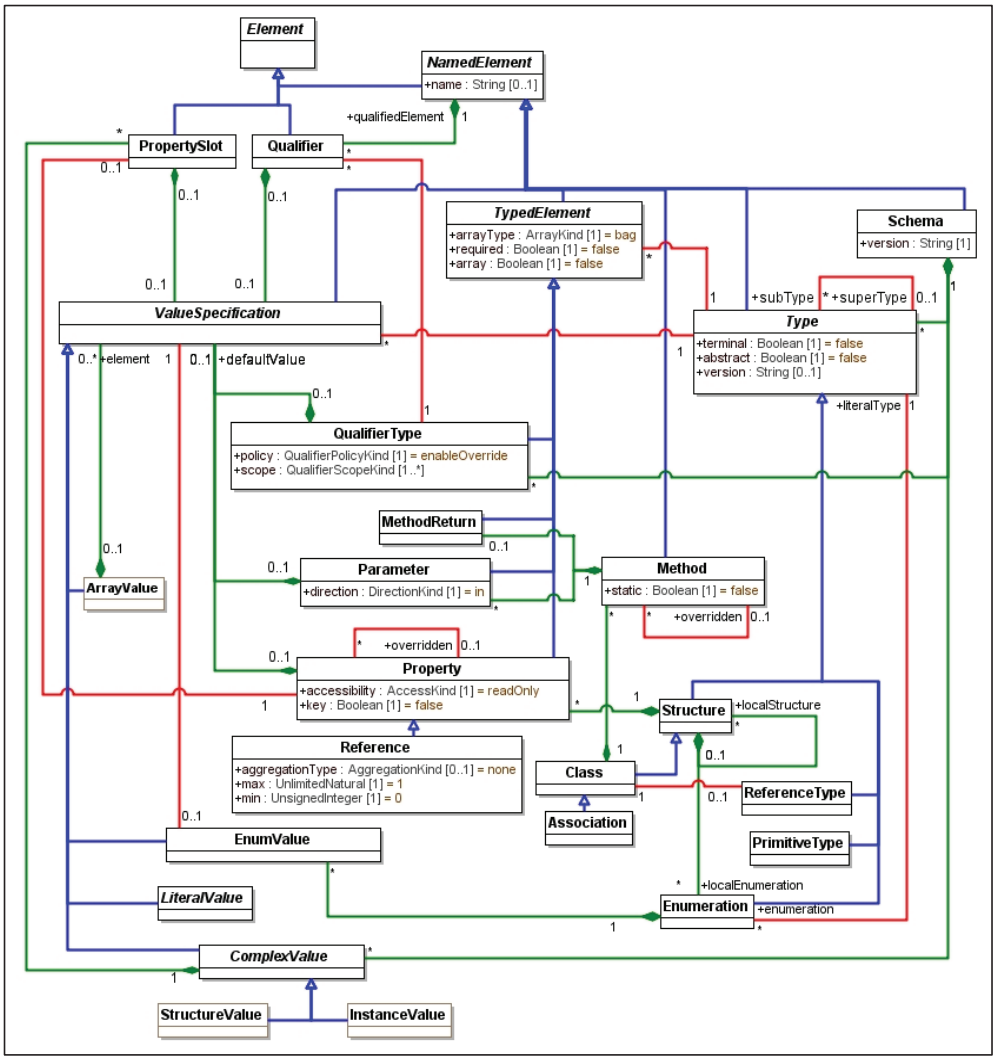


Abbildung 4.12: CIM Metamodel [Dist14]

CIM ist eine sehr komplexe hierarchische Sprache, in der mittlerweile über 700 Klassen beschrieben sind, was den Nachteil hat, dass die Übersichtlichkeit und Nachvollziehbarkeit leidet [KeKS01]. Zudem liegt der Fokus sehr stark auf der Modellierung von Infrastrukturen und weniger auf der Modellierung von IT-Services. TOSCA bietet eine übersichtlichere Darstellung von IT-Services, wobei zur Beschreibung von Infrastrukturkomponenten bei Bedarf auch auf CIM-Modelle zurückgegriffen werden kann (vgl. [Dist12b, Oasi13a]).

4.3.2 Topology and Orchestration Specification for Cloud Applications (TOSCA)

TOSCA ist ein von OASIS entwickelter Standard, um die Portabilität von Cloud-Anwendungen zwischen verschiedenen Cloud-Anbietern sowie die Automatisierung des Deployments von Cloud-Anwendungen zu unterstützen [Oasi13a], [Oasi13b]. Dazu wird eine auf XML Schema [Oasi13c] basierte Modellierungssprache verwendet, um den Aufbau von IT-Services, in Form von Service Templates zu beschreiben. Ein Service Template besteht aus einem Topology Template, Node Templates, Relationship Templates und Plänen. Node und Relationship Templates werden durch Node- bzw. Relationship Types typisiert. Abbildung 4.13 stellt ein TOSCA Service Template graphisch da.

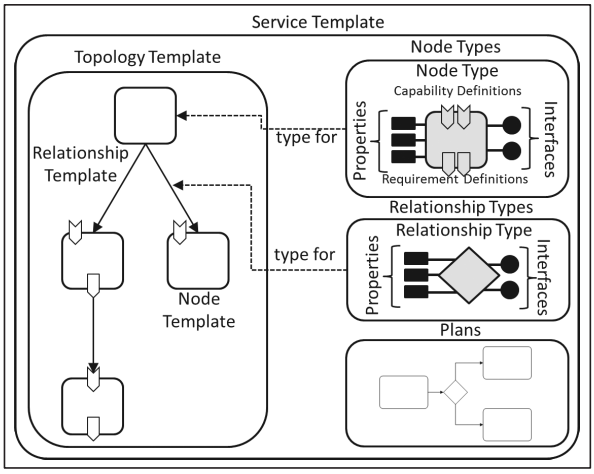


Abbildung 4.13: TOSCA Service Template [Oasi13a]

Ein Topology Template beschreibt die Struktur von IT-Service-Komponenten als einen gerichteten Graphen und besteht aus Node Templates, die mittels Relationship Templates miteinander verbunden sind.

Ein Node Template repräsentiert eine IT-Service-Komponente, z.B. einen Application Server und wird durch einen Node Type typisiert. Node Types können von anderen Node Types erben und definieren für die jeweiligen Node Templates die

- Eigenschaften der Komponente (*PropertiesDefinition*),
- verfügbare Operationen zur Manipulation der Komponente (*Interfaces*),
- Anforderungen der Komponente (*RequirementDefinitions*),

- möglichen Status einer Komponente (*InstanceStates*),
- Ressourcen, die eine Komponente zur Erfüllung der Anforderungen einer anderen Komponente bereitstellt (*CapabilityDefinitions*).

Die im Node Type mit konkreten Werten definierten Status und verfügbaren Operationen einer Komponente gelten für alle instanziierten Node Templates dieses Node Types und werden daher in Node Templates nicht extra definiert. Für die Eigenschaften, Anforderungen und Ressourcen eines Node Templates, wird innerhalb eines Node Types jedoch nur die Struktur festgelegt, so dass innerhalb eines Node Templates

- konkrete Werte für Eigenschaften (*Properties*),
- Anforderungen (*Requirements*) und
- Ressourcen, die anderen Komponenten zur Verfügung gestellt werden (*Capabilities*), angegeben werden müssen. Zusätzlich können in Node Templates
- die Anzahl der möglichen Instanzen eines Node Templates eingeschränkt werden (*minInstances*, *maxInstances*),
- Restriktionen für Eigenschaften festgelegt werden (*PropertyConstraints*),
- sowie *Policies* und
- *DeploymentArtifacts* hinterlegt werden.

DeploymentArtifacts ermöglichen es, Skripte und andere Software-Artefakte innerhalb eines Node Templates zu hinterlegen, um die modellierten Cloud-Anwendungen in einer entsprechenden TOSCA kompatiblen Umgebung automatisiert deployen zu können [BBHK13].

Relationship Templates werden durch Relationship Types typisiert und beschreiben die Beziehung zwischen Node Templates, beispielsweise, wenn eine IT-Service-Komponente auf einer anderen gehostet wird, oder zwei IT-Service-Komponenten miteinander kommunizieren. Abbildung 4.14 verdeutlicht die verschiedenen konzeptuellen Schichten von TOSCA.

Die Ausführung von in Node Templates hinterlegten Software-Artefakten wird durch Pläne orchestriert. Pläne sind Ablaufmodelle, die zur Orchestrierung der Operationen, die unter *Interfaces* eines Node Templates beschrieben sind, eingesetzt werden. In den TOSCA Spezifikationen werden keine Vorgaben bezüglich einer zu nutzenden Modellierungssprache gemacht, so dass Pläne in verschiedenen Modellierungssprachen beschrieben werden können, beispielsweise mittels XML-Netzen.

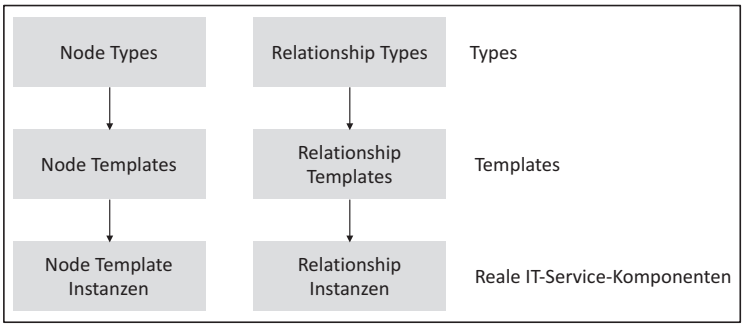


Abbildung 4.14: Konzeptuelle Schichten von TOSCA (angelehnt an [BBKL14])

Abbildung 4.15 zeigt beispielhaft die Topologie der Anwendung „MyApplication“. Quelltext 4.1 beschreibt den Node Type „Application“ im XML-Format, welches das Node Template für die Anwendung „MyApplication“ typisiert. Quelltext 4.2 zeigt exemplarisch das Node Template für „MyApplication“ im XML-Format. Auf die Verwendung von Namensräumen wurde aus Gründen der Lesbarkeit verzichtet. Für ein vollständiges Beispiel eines Service Templates für die Anwendung SugarCRM siehe [Thom16].

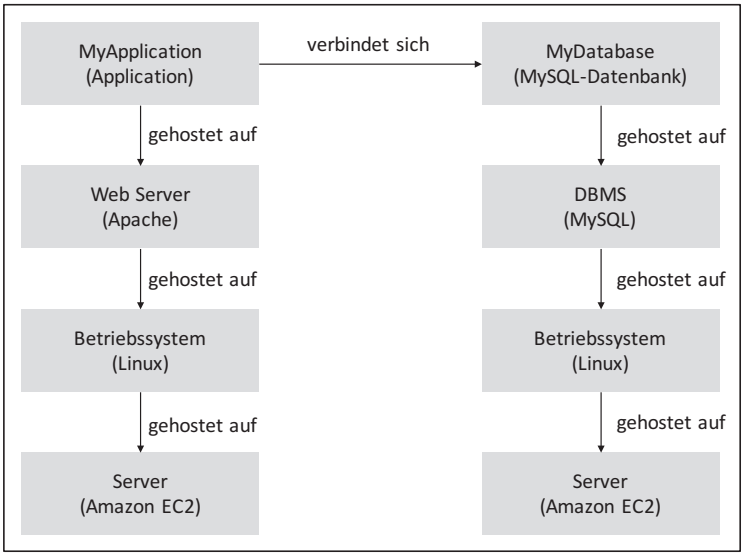


Abbildung 4.15: Beispielhafte Topologie einer Anwendung

```
1  <NodeType name="Application">
2   <PropertiesDefinition>
3    <Property element="User"/>
4    <Property element="Password"/>
5    <Property element="DBConnection"/>
6   </PropertiesDefinition>
7   <RequirementDefinitions>
8    <RequirementDefinition name="Database" requirementType="DatabaseType"/>
9    <RequirementDefinition name="WebServer" requirementType="Host"/>
10  </RequirementDefinitions>
11  <CapabilitiyDefinitions>
12   <CapabilityDefinition name="WebService" requirementType="WebService-
      Type“/>
13  </CapabilitiyDefinitions>
14  <InstanceStates>
15   <InstanceState state="available"/>
16   <InstanceState state="started"/>
17   <InstanceState state="stopped"/>
18   <InstanceState state="unavailable"/>
19  </InstanceStates>
20  <Interfaces>
21   <Interface name="Operations">
22    <Operation name="deploy"/>
23    <Operation name="start"/>
24    <Operation name="stop"/>
25    <Operation name="configure"/>
26    <Operation name="undeploy"/>
27   </Interface>
28  </Interfaces>
29 </NodeType>
```

Quelltext 4.1: Node Type „Application“

```
1  <NodeTemplate id="MyApplicationID" name="MyApplication" nodeType="Applica-
   tion">
2   <Properties>
3    <User>admin</User>
4    <Password>admin</Password>
5    <DBConnection>jdbc:mysql://localhost:1521/XE</DBConnection>
6   </Properties>
7   <Requirements>
8    <Database>MyOwnDatabase</Database>
9    <WebServer>MyTomcat</WebServer>
10  </Requirements>
11  <Capabilities>
12   <WebService>MyWebservice</WebService>
13  </Capabilities>
14 </NodeTemplate>
```

Quelltext 4.2: Node Template "MyApplication"

TOSCA ist mit dem Ziel entstanden Cloud-Anwendungen über den gesamten Lebenslauf mithilfe von TOSCA-Modellen zu verwalten. Um Cloud-Anwendungen über das initiale Deployment hinaus mithilfe von TOSCA zu steuern, ist hierfür eine Repräsentation der tatsächlichen Instanzen einer Cloud-Anwendung notwendig. Die im aktuellen Standard beschriebenen Service Templates erlauben jedoch nur das Deployment dieser und bieten keine Möglichkeit die zur Laufzeit gewonnen Informationen der tatsächlich erzeugten Instanzen in einem Modell zu speichern. Zum Beispiel kann in einem Service Template definiert werden, dass zwei Datenbankserver erstellt werden müssen und diese eine IP in einem bestimmten Subnetz erhalten. Es wird jedoch keine feste IP-Adresse definiert, sondern diese kann zur Laufzeit zufällig gewählt werden. Wenn dieses Service Template nun genutzt wird, um die Datenbankserver zu erzeugen, können die vergebenen IP-Adressen nicht aus dem Service Template ermittelt werden. Zur Lösung dieses Problems wurde ein erster Vorschlag für ein Instanzenmodell erstellt, welches sich noch in Abstimmung befindet. Zum Zeitpunkt der Erstellung dieser Arbeit wurde dieses noch nicht in den Standard übernommen [Oasi17].

Eine weitere Einschränkung von TOSCA ist, dass in TOSCA Node Templates unabhängig von Plänen erstellt werden. Das führt dazu, dass keine integrierte Modellierung zwischen Node Templates und Plänen unterstützt wird. Die Erstellung von Plänen selbst wird sehr vage beschrieben, so dass keine Richtlinien und konkrete Beschreibungen für den Einsatz von Modellierungssprachen gegeben werden. Aus diesem Grund haben Kopp et al. [KBBL12]

eine Erweiterung von BPMN für den Einsatz mit TOSCA entwickelt. Eine integrierte Modellierung von BPMN-Plänen mit TOSCA ist jedoch auch mit dieser Erweiterung nicht möglich.

Für die Erkennung von Fehlern wird allerdings ein integriertes Modell benötigt, um Pattern zur Fehlererkennung aus dem Modell instanziieren zu können. Im Folgenden wird daher eine Methode vorgestellt, die auf Basis von TOSCA und XML-Netzen eine integrierte Modellierung von IT-Service-Wartungen ermöglicht.

4.4 Modellierung von IT-Service-Wartungen

Um mithilfe eines Modells, wie in Kapitel 3.5 beschrieben, Fehler erkennen zu können, wird eine Repräsentation der tatsächlichen IT-Service-Wartung benötigt. Dazu werden in dieser Arbeit folgende Anforderungen an ein solches Modell gestellt:

- Die Operationen, die auf einer IT-Service-Komponente ausgeführt werden können, müssen modelliert werden können.
- Der zu wartende IT-Service und seine IT-Service-Komponenten, müssen modelliert und eindeutig identifiziert werden können.
- Konfigurationsänderungen müssen repräsentiert werden, d.h. es müssen Konfigurationsparameter und die entsprechenden Konfigurationswerte modelliert werden können.
- Der Zustand einer IT-Service-Komponente, und in welchem Zustand sich diese für die Durchführung einer Operation befinden muss, muss modelliert werden.

Wie in Kapitel 4.3 beschrieben, wird zur Modellierung von sich in Betrieb befindlichen IT-Services eine deskriptive Modellierungsperspektive, wie sie CIM bietet, eingesetzt. Nachfolgend soll jedoch bewusst nicht auf Basis von CIM eine IT-Service-Wartung modelliert werden, sondern auf Basis von TOSCA. TOSCA bietet durch die automatisierte Bereitstellung von IT-Services das Potenzial, Laufzeitinformationen in einer entsprechenden TOSCA-kompatiblen Umgebung, automatisiert zu erfassen, so dass diese nicht manuell gepflegt werden müssen. Ein Beispiel hierfür ist die Zuweisung einer IP-Adresse während der Bereitstellung, die daraufhin durch die Laufzeitumgebung im Node Template gesichert wird. Da der aktuelle Standard von TOSCA noch keine explizite Modellierung von IT-Service-Instanzen beinhaltet, werden in dieser Arbeit folgende Änderungen an der TOSCA-Spezifikation vorgenommen:

- Ein Node Template repräsentiert immer exakt eine Instanz einer IT-Service-Komponente, so dass für alle Node Templates gilt *minInstances*=1 und *maxInstances*=1.

- Node Templates erhalten zusätzlich das komplexe Element *InstanceState* mit dem Unterelement *State,* das den aktuellen Status der korrespondierenden IT-Service-Komponente beschreibt. Es sind nur Status erlaubt, die in *InstanceStates* des korrespondierenden Node Types beschrieben sind. Der Status eines Node Templates kann nach der initialen Bereitstellung nur durch eine IT-Service-Wartung geändert werden, d.h. Störungen einer IT-Service-Komponente im laufenden Betrieb, die zu einem Ausfall dieser Komponenten führen, werden nicht im Node Template repräsentiert.

Im TOSCA XML Schema werden hierzu, wie im Quelltext 4.3 abgebildet, die Zeilen 19-24 und 56-62 hinzugefügt.

```
1   <xs:complexType name="tNodeTemplate">
2    <xs:complexContent>
3     <xs:extension base="tEntityTemplate">
4      <xs:sequence>
5       <xs:element name="Requirements" minOccurs="0">
6        <xs:complexType>
7         <xs:sequence>
8          <xs:element name="Requirement" type="tRequirement" maxOccurs="un-
           bounded"/>
9         </xs:sequence>
10       </xs:complexType>
11      </xs:element>
12      <xs:element name="Capabilities" minOccurs="0">
13       <xs:complexType>
14        <xs:sequence>
15         <xs:element name="Capability" type="tCapability" maxOccurs="unbounded"/>
16        </xs:sequence>
17       </xs:complexType>
18      </xs:element>
19      <xs:element name="InstanceState">
20         <xs:complexType>
21            <xs:sequence>
22               <xs:element name="State" type="tState"/>
23            </xs:sequence>
24         </xs:complexType>
25      </xs:element>
26      <xs:element name="Policies" minOccurs="0">
```

```
27      <xs:complexType>
28       <xs:sequence>
29        <xs:element name="Policy" type="tPolicy" maxOccurs="unbounded"/>
30       </xs:sequence>
31      </xs:complexType>
32     </xs:element>
33     <xs:element name="DeploymentArtifacts" type="tDeploymentArtifacts"
       minOccurs="0"/>
34    </xs:sequence>
35    <xs:attribute name="name" type="xs:string" use="optional"/>
36    <xs:attribute name="minInstances" type="xs:int" use="optional" default="1"/>
37    <xs:attribute name="maxInstances" use="optional" default="1">
38     <xs:simpleType>
39      <xs:union>
40       <xs:simpleType>
41        <xs:restriction base="xs:nonNegativeInteger">
42         <xs:pattern value="([1-9]+[0-9]*)"/>
43        </xs:restriction>
44       </xs:simpleType>
45       <xs:simpleType>
46        <xs:restriction base="xs:string">
47         <xs:enumeration value="unbounded"/>
48        </xs:restriction>
49       </xs:simpleType>
50      </xs:union>
51     </xs:simpleType>
52    </xs:attribute>
53   </xs:extension>
54  </xs:complexContent>
55 </xs:complexType>
56 <xs:complexType name="tState">
57    <xs:complexContent>
58       <xs:extension base="tEntityTemplate">
59          <xs:attribute name="name" type="xs:string" use="required"/>
60       </xs:extension>
61    </xs:complexContent>
62 </xs:complexType>
```

Quelltext 4.3: Ausschnitt TOSCA XML Schema

Zur Beschreibung von IT-Service-Wartungen werden nachfolgend XML-Netze zur Modellierung des Wartungsablaufs und TOSCA zur Modellierung der zu wartenden IT-Service-Komponenten eingesetzt. Diese neue Form von Netzen wird im weiteren Verlauf der Arbeit als Wartungsplan bezeichnet. Hierfür werden in Anlehnung an [Vett16] folgende Anpassungen an der allgemeinen Definition von XML-Netzen vorgenommen:

- Alle Stellen sind mit dem TOSCA XML Schema typisiert, d.h. alle Stellen dienen als Container für Service Templates.
- Zusätzlich ist jeder aus einer Transition ausgehenden Kante ein Node Type zugeordnet. Das Node Type schränkt die möglichen Operationen der Transition und die möglichen Filterschema immer auf die entsprechenden Node Templates im Service Template ein.
- Mit Filterschemata können entweder Node Templates ausgewählt werden, oder der Status bzw. die Eigenschaften eines Node Templates geändert werden. Das Löschen von ganzen Node Templates ist nicht möglich. Stattdessen wird der Status eines Node Templates geändert, beispielsweise von „deployed“ zu „undeployed“. Der Grund hierfür ist, dass auch IT-Service-Komponenten im Status „undeployed“ bis zum Ende der Wartung überwacht werden müssen, um sicherzugehen, dass diese nicht aus Versehen wieder deployed wurden. Das Löschen von Eigenschaften bzw. Eigenschaftswerten eines Node Templates ist hingegen erlaubt, beispielsweise wenn ein User entfernt wird.
- Das Manipulieren von Node Templates oder das Löschen von Eigenschaften bzw. Eigenschaftswerten eines Node Templates ist nur mit Filterschematas möglich, die einer ausgehenden Kante einer Transition zugeordnet sind. Ausgehenden Kanten einer Stelle können nur lesende Filterschemata zugeordnet werden.
- Transitionen repräsentieren Operationen auf Node Templates. Es sind nur Operationen gültig, die unter *Interfaces* der Node Types enthalten sind, die die Stellen im Nachbereich der Transition typisieren.
- Transitionen enthalten zusätzlich die beiden Attribute *Start* und *Ende*. Diesen geben an von wann bis wann eine Transition ausgeführt werden darf.

Der Zeitraum vom geplanten Start bis zum geplanten Ende einer IT-Service-Wartung wird nachfolgend als Wartungsfenster bezeichnet. Formal wird ein Wartungsplan wie folgt definiert:

Definition 4.10: Wartungsplan

Ein Wartungsplan ist ein Tupel $WP = (S, T, F, \Psi, ST, FI, KB, TI, M^0)$, für das gilt:

(i) (S, T, F) ist ein Petri-Netz.
(ii) $\Psi = (D, FU, PR)$ ist eine Struktur, die aus endlichen Individuenmengen D, einer Menge FU von auf D definierten Funktionen, sowie einer Menge PR von auf D definierten Prädikaten mit unveränderlichen Ausprägungen besteht.
(iii) Die Funktion ST: $S \rightarrow TOSCA\ XMl\ Schema$ weist allen Stellen das TOSCA XML Schema zu.
(iv) Die Funktion FI: $F \rightarrow NT$ weist zusätzlich jeder von einer Transition ausgehenden Kante ein Node Type zu.
(v) KB beschriftet Kanten aus F mit einer Menge valider, bezüglich des TOSCA XML Schemas und des Node Types zulässiger Filterschemata.
(vi) TI weist jeder Transition $t \in T$ einen in Ψ gebildeten prädikatenlogischen Ausdruck als Inschrift zu. Jede Variable in diesem Ausdruck muss in der Beschriftung einer zur Transition adjazenten Kante vorkommen.
(vii) M markiert die Stellen $s \in S$ mit einem, bezüglich des TOSCA XML Schemas gültigen, Service Templates. M^0 ist die Startmarkierung.
(viii) Jede Transition $t \in T$ repräsentiert genau eine *Operation* aus der Menge
(ix) *Interfaces* der Node Types, die den aus der Transition ausgehenden Kanten zugeordnet sind.
(x) Jede Transition $t \in T$ enthält die beiden Attribute *Start* und *Ende*.
(xi) Ein Wartungsplan ist stellenberandet.

■

Abbildung 4.16[2] zeigt einen beispielhaften Wartungsplan, in dem die Datenbankverbindung der IT-Service-Komponente MYAPPLICATION konfiguriert wird. Für diese Wartung werden genau zwei Node Types APPLICATION und DATABASE benötigt[3]. Die Wartung startet mit dem Lesen des Service Templates, unter der Bedingung, dass alle Node Templates des Typs APPLICATION im Status STARTED sind. Anschließend wird mittels des Filterschematas FS1 MYAPPLICATION (NT1) in den Status STOPPED versetzt. Dazu wird die Transition STOP geschaltet, die die entsprechende Operation des Node Types APPLICATION repräsentiert. Anschließend wird die Datenbankverbindung auf Basis der Properties der Datenbank

[2] Aus Gründen der Lesbarkeit werden die Filterschemata informal beschrieben.
[3] In der Abbildung sind die Node Templates abgebildet, die ein Node Type instanziieren. Das Node Type APPLICATION ist dem Quelltext 4.1 zu entnehmen. Das Node Type DATABASE ist dem Quelltext 4.4 zu entnehmen.

MYDATABASE (NT2) mittels des Filterschematas FS2 konfiguriert. Daraufhin wird MYAPPLICATION, unter der Bedingung, dass MYDATABASE (NT2) gestartet ist, und MYAPPLICATION (NT3) gestoppt ist, gestartet.

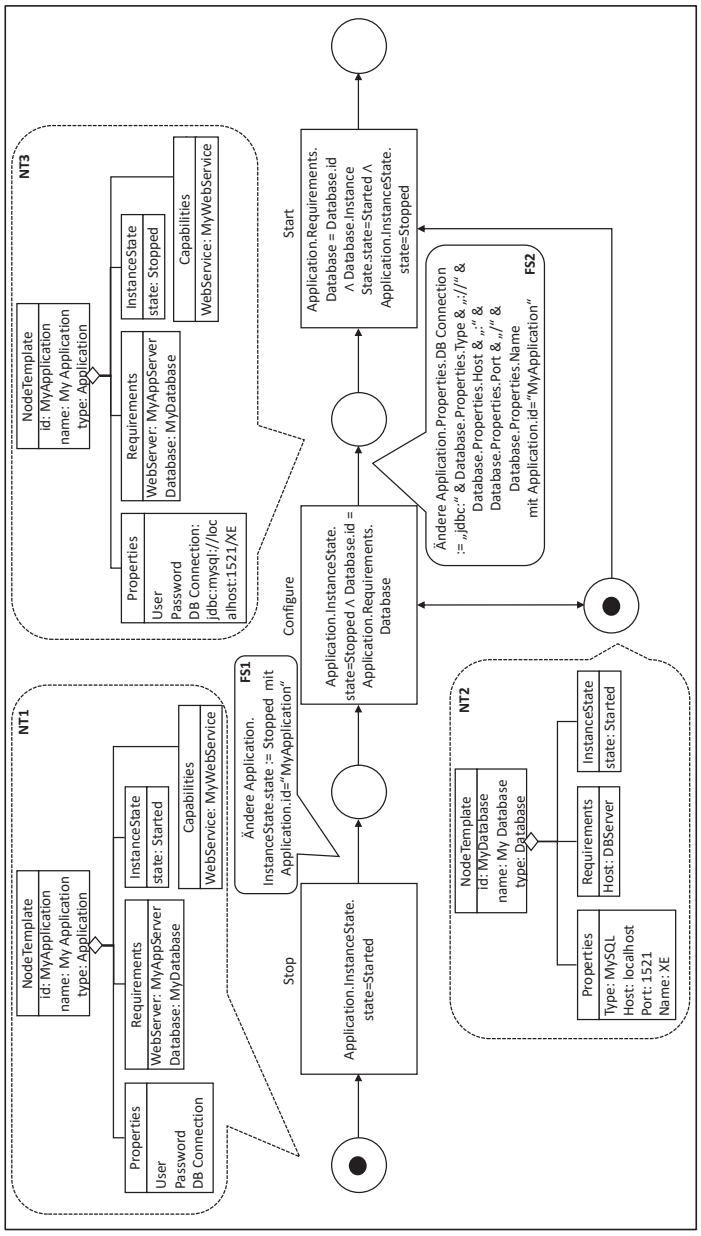


Abbildung 4.16: Beispielhafter Wartungsplan (angelehnt an [Vett16])

```
1   <NodeType name=„Database“>
2    <PropertiesDefinition>
3     <Property element=„Type“/>
4     <Property element=„Host“/>
5     <Property element=„Port“/>
6     <Property element=„Name“/>
7    </PropertiesDefinition>
8     <RequirementDefinitions>
9     <RequirementDefinition name=„WebServer“ requirementType="Host"/>
10   </RequirementDefinitions>
11  <InstanceStates>
12   <InstanceState state="available"/>
13   <InstanceState state="started"/>
14   <InstanceState state="stopped"/>
15   <InstanceState state="unavailable"/>
16   </InstanceStates>
17   <Interfaces>
18   <Interface name=„Operations“>
19    <Operation name=„install“/>
20    <Operation name="start"/>
21    <Operation name=„shutdown“/>
22    <Operation name=„uninstall“/>
23   </Interface>
24   </Interfaces>
25  </NodeType>
```

Quelltext 4.4: Node Type "Database"

4.5 Zusammenfassung des Kapitels

In diesem Kapitel wurden die Grundlagen zur Modellierung von betrieblichen Abläufen und IT-Service-Komponenten vorgestellt. Die Modellierung von betrieblichen Abläufen wurde anhand von Petri-Netzen beschrieben. Nach der Dokumentation der elementaren Elemente eines Petri-Netzes, wurden einfache Petri-Netze und elementare Schaltmuster vorgestellt. Um auch objektbezogene Aspekte in Ablaufmodellen berücksichtigen zu können, können höhere Petri-Netze verwendet werden. XML-Netze sind eine Variante höherer Petri-Netze, in denen Stellen als Container für XML-Dokumente dienen. Mit XML-Netzen ist

es somit möglich die Manipulation von XML-Dokumenten zu modellieren. Dies ist von Vorteil, da IT-Services bzw. die IT-Service-Komponenten ebenfalls in Form von XML-Dokumenten beschrieben werden können. Als Vertreter von Modellierungssprachen zur Modellierung von IT-Service-Komponenten wurden die Sprachen CIM und TOSCA vorgestellt. CIM ist eine umfangreiche deskriptive Modellierungssprache, die es erlaubt IT-Service-Komponenten, insbesondere Infrastruktur-Komponenten zu beschreiben und hierfür eine umfassende Sammlung von Klassen bereitstellt. Aufgrund des Umfangs hat CIM jedoch den Nachteil, dass die Übersichtlichkeit und Nachvollziehbarkeit leidet. TOSCA ist eine Modellierungssprache, die einen stärkeren Fokus auf die Beschreibung von IT-Services legt und sowohl die Struktur der IT-Service-Komponenten beschreibt, als auch die Modellierung von Abläufen zum Management der IT-Service-Komponenten vorsieht. Aufgrund der stärkeren Abstrahierung von Infrastrukturkomponenten ist TOSCA zudem übersichtlicher als CIM. Da TOSCA durch ein XML Schema definiert ist, eignet es sich außerdem sehr gut für den Einsatz mit XML-Netzen. Wie IT-Service-Wartungen mit XML-Netzen und TOSCA modelliert werden können, wurde zum Schluss des Kapitels beschrieben. Diese Form der Modellierung von IT-Service-Wartungen hat den Vorteil, dass sowohl der Ablauf der IT-Service-Wartung, als auch die notwendigen Konfigurationen von IT-Service-Komponenten in einem Modell beschrieben werden können. Ein solches Modell wird als Wartungsplan bezeichnet und dient als Grundlage für die Ableitung der für die Fehlererkennung benötigten Pattern-Instanzen. Die in dieser Arbeit verwendeten Pattern werden in nachfolgendem Kapitel vorgestellt.

5 Pattern und Anti-Pattern

Um Fehler bei der Durchführung von IT-Service-Wartungen zu erkennen, werden in dieser Arbeit Pattern bzw. Anti-Pattern eingesetzt. Es sei nochmals darauf hingewiesen, dass in der Design Phase eines Wartungsplans Pattern zum Einsatz kommen, die den korrekten Ablauf der IT-Service-Wartung beschreiben. Bei der Überprüfung der Durchführung der IT-Service-Wartung zur Laufzeit sind jedoch Anti-Pattern erforderlich, da die Prüfeinheit nur bei einer Nicht-Einhaltung eines solchen Pattern eine Meldung erzeugen soll und nicht, wenn der Ablauf gemäß des Patterns erfolgt ist. Um alle zur Kontrolle einer Wartungsdurchführung benötigten Anti-Pattern abzuleiten, müssen zunächst alle relevanten Pattern aus dem Wartungsplan, welcher die IT-Service-Wartung beschreibt, identifiziert werden. Dazu wird nachfolgend ein Überblick über bekannte Pattern im Forschungsgebiet der Geschäftsprozessmodellierung gegeben, um darauf aufbauend zu überprüfen, welche Pattern sich für den Einsatz in IT-Service-Wartungen eignen. Zunächst wird jedoch näher auf den Begriff Pattern eingegangen.

5.1 Der Begriff Pattern

In der Informatik hat das Konzept des Patterns seit dem Erscheinen des Buchs "Design Pattern: Elements of Reusable Object-Oriented Software" von Gamma et al. [GHJV94] im Jahr 1994 an Bedeutung gewonnen. In diesem Buch beschreiben Gamma et al. Design Pattern, die sie bei der Anwendung objektorientierter Programmiersprachen identifiziert haben. Design Pattern ermöglichen es, von tatsächlichen Programmiersprachen zu abstrahieren, um so Lösungen zu präsentieren, die unabhängig von der eingesetzten Programmiersprache und Software sind. Aalst et al. [AHKB03] haben das Konzept der Design Pattern auf die Ablaufmodellierung übertragen und im Jahr 2003 zunächst für die Ablauf-Perspektive von Ablaufmodellen 20 Pattern vorgestellt. Mittlerweile wurden weitere Pattern beschrieben, so dass sich die Anzahl an Pattern für die Ablauf-Perspektive auf über 40 erhöht hat [ATRD15]. Zusätzlich wurden Pattern für weitere Perspektiven eines Ablaufmodells, wie zum Beispiel für die Ressourcen-Perspektive [RAHE05], die Daten-Perspektive [RHEA05] oder die Zeit-Perspektive [LaWR14] vorgestellt. Eine Übersicht bekannter Pattern für die Ablaufmodellierung wird in Kapitel 5.2 gegeben.

Gamma et al. [GHJV94] definieren ein Pattern als eine Lösung für ein wiederkehrendes Problem in einem bestimmten Kontext. Dieses Verständnis eines Patterns ist sehr weit verbreitet [RiZü96] und baut auf der ursprünglichen Definition des Architekten Christopher Alexander auf, der ein Pattern als „three part rule, which expresses a relation between a certain context, a problem, and a solution“ [Alex79, S.247] beschreibt. Dieses Verständnis eines Patterns richtet den Fokus auf die Erstellung eines Artefakts zur Lösung eines Problems. Riehle und Züllighoven [RiZü96] definieren ein Pattern allgemeiner als „abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts“. Demnach sind Pattern Konzepte, die Menschen aus ihrer Erfahrung heraus entwerfen. Aufgrund dieser Erfahrung und der Fähigkeit Erfahrungen zu reflektieren, sind Menschen in der Lage, Instanzen eines Pattern wiederzuerkennen, aber auch selbst zu erstellen. Pattern werden somit nicht nur für die Erstellung von Artefakten eingesetzt, sondern dienen auch zur Analyse, um Instanzen davon zu erkennen [RiZü96]. Ein Beispiel für die Verwendung eines Pattern zur Analyse in einem bestimmten Kontext ist die Verwendung von bestimmten Werkzeugen für die Bearbeitung eines Materials, wie ein Füller zum Ausfüllen eines Papierformulars. Aufgrund unserer Erfahrungen wissen wir, dass der Füller zum Ausfüllen eines Papierformulars verwendet werden kann und nicht beispielsweise eine Büroklammer, die zum Zusammenheften von Papierformularen verwendet wird [RiZü96]. Ein Beispiel für die Erstellung einer Instanz eines Patterns ist die Herstellung eines Füllers. Aufgrund des allgemeinen Designs eines Füllers, das Design-Pattern, können Instanzen eines Füllers hergestellt werden. Dieses allgemeine Verständnis eines Patterns wird in dieser Arbeit geteilt und ein Pattern daher nach Riehle und Züllighoven [RiZü96] wie folgt definiert:

Definition 5.1: Pattern

Ein Pattern ist eine Abstraktion einer erwünschten, konkreten, wiederkehrenden Form in einem bestimmten Kontext.

■

Unter einer Form wird eine begrenzte Anzahl unterscheidbarer Einheiten und ihrer Beziehungen untereinander verstanden. Einheiten können technische Einheiten, beispielsweise ein maschineller Aufgabenträger, aber auch nicht technische Einheiten sein, beispielsweise Aktivitäten in einem betrieblichen Prozess, die in einer Beziehung zueinander stehen wie, dass Aktivität A ausgeführt sein muss, bevor Aktivität B ausgeführt werden kann [RiZü96].

Die Form eines Patterns ist wiederum vom jeweiligen Kontext abhängig, bzw. erhält seine Form erst durch den Kontext, und eignet sich aufgrund dessen speziell für den Einsatz in diesem spezifischen Kontext. Der Kontext, als auch die Form, sind ein Konstrukt, das

Menschen aufgrund ihrer Erfahrung erstellen. Wird ein Pattern in einem neuen Kontext angewendet, können damit notwendige Anpassungen am Pattern einhergehen. Wird der Kontext geändert, muss die Form somit ebenfalls geändert werden und vice versa.

Ein Pattern ist generell die Abstraktion einer erwünschten Form, d.h. ein Pattern spiegelt eine Form, wie es Menschen planen, bzw. für gut befinden. Ein Anti-Pattern ist hingegen die Abstraktion einer unerwünschten Form, bzw. einer Form, die nicht geplant ist oder auftreten sollte. Entsprechend wird ein Anti-Pattern wie folgt definiert:

Definition 5.2: Anti-Pattern

Ein Anti-Pattern ist eine Abstraktion einer unerwünschten, konkreten, wiederkehrenden Form in einem bestimmten Kontext.

■

Nachfolgend werden Pattern im Kontext betrieblicher Prozesse vorgestellt, um darauf aufbauend zu untersuchen, welche Pattern sich speziell zur Erkennung von Fehlern bei der Durchführung von IT-Service-Wartungen eignen.

5.2 Pattern-Übersicht

In einer systematischen Übersichtsarbeit haben Fellmann et al. [FKLS18] Pattern im Kontext von betrieblichen Prozessen untersucht und in folgende neun Bereiche unterteilt:

- Struktur- und Ablauf-Pattern
  Diese Pattern beschreiben grundlegende Konstrukte zur Modellierung von Abläufen. Unter anderem sind die von Aalst et al. [AHKB03] vorgestellten Workflow Pattern dieser Kategorie zugeordnet. In dieser Kategorie werden Pattern weiter unterteilt in die Kategorien *Instanziierung und Terminierung*, sowie *Prozessstruktur und -ablauf*.
- Ressourcen-Pattern
  Diese Pattern beschreiben Ressourcen, die an einem Prozess beteiligt sind. Die Kategorie wird weiter unterteilt in die Unterkategorien *Rollen und Ressourcen*, sowie *Delegation*.
- Daten-Pattern
  Diese Pattern beschreiben Daten, die in einem Prozess konsumiert, erzeugt oder weitergeleitet werden.

- Orchestrierung- und Choreografie-Pattern
  Diese Kategorie enthält Pattern, die die Organisation personeller Aufgabenträger (Choreografie) oder maschineller Aufgabenträger (Orchestrierung) zur Ausführung von Aktivitäten in Prozessen beschreiben. Es werden die drei Unterkategorien *Kommunikation*, *Verhandlung* und *Kollaboration* unterschieden. Die Unterkategorie *Kommunikation* enthält dabei Pattern, die den Austausch von Daten zwischen maschinellen Aufgabenträgern zur Ausführung von Aktivitäten in einem betrieblichen Prozess beschreiben. Die Unterkategorien *Verhandlung* und *Kollaboration* beschreiben die Verhandlung von Verträgen, wie ein betrieblicher Prozess ausgeführt werden soll, und die Organisation der kollaborativen Arbeit zwischen personellen Aufgabenträgern.
- Content-Pattern
  In dieser Kategorie werden Content-Pattern zusammengefasst. Darunter werden Pattern verstanden, die die semantische Bedeutung von Prozessfragmenten adressieren, um beispielsweise bestimmte betriebswirtschaftlichen Ziele zu erreichen. Diese Pattern werden nochmals untergliedert in *Domänen-abhängige* Pattern, z.B. für die Domänen Gesundheitswesen und Softwareentwicklung, sowie *Domänen-unabhängige* Pattern, die für alle Domänen gültig sind. Beispiele für Content-Pattern sind die in der betrieblichen Praxis domänenunabhängig auftretenden Prozessfragmente „Überprüfung“ oder „Genehmigung“.
- Qualität-, Compliance- und Risiko-Pattern
  Diese Pattern haben zum Ziel die Einhaltung, von Qualitätszielen und regulatorischen Anforderungen in betrieblichen Prozessen sicherzustellen. Die Pattern dieser Kategorie werden eingeteilt in die Unterkategorien *Compliance*, *Risiko und Sicherheit*, sowie *Umwelteinfluss*. Pattern in der Unterkategorie Compliance dienen zur Berücksichtigung von regulatorischen Anforderungen, beispielsweise die Einhaltung einer bestimmten Reihenfolge durchzuführender Aktivitäten, oder zeitlicher Restriktionen. Compliance-Pattern beschreiben somit Abläufe, die bei der eigentlichen Prozessausführung eingehalten werden müssen [BeKl14].
- Adaptierung- und Verbesserung-Pattern
  In dieser Pattern-Kategorie werden Pattern zusammengefasst, die die Beschreibung der Variabilität von Prozessen hinsichtlich möglicher Erweiterungen oder Anpassungen zum Ziel haben. Dabei wird zwischen den Unterkategorien *Unterschiede und Variabilität* und *Modellverbesserungen* unterschieden. In der Unterkategorie *Unterschiede und Variabilität* werden Pattern zusammengefasst, um beispielsweise die Flexibilität eines Prozesses zur Laufzeit zu beschreiben. Pattern in der Kategorie

Modellverbesserungen beschreiben Konstrukte zur Verbesserung existierender Prozesse, beispielsweise um die Lesbarkeit zu verbessern.

- Integration- und Transformation-Pattern
  In dieser Kategorie werden Pattern zusammengefasst, die eine Transformation oder Integration von Prozessmodellen in andere Modelle oder zwischen verschiedenen Modellierungssprachen bzw. Modellierungsperspektiven beschreiben. Es wird zwischen den vier Unterkategorien *Modellperspektiven*, *Modellintegration*, *Modelltransformation* und *Anwendung von Modellen und der Modellierung* unterschieden.
- Prozessarchitektur-Pattern
  Diese Pattern beschreiben die Beziehungen zwischen verschiedenen Komponenten einer Unternehmensarchitektur, z.B. Geschäftspläne, Kunden, Anforderungen, oder Infrastrukturkomponenten, in einer Prozessarchitektur. Diese Kategorie wird nicht in weitere Unterkategorien aufgeteilt.

Diese verschiedenen Pattern-Kategorien sind nicht als vollständig zu betrachten, sondern stellen eine Bestandsaufnahme bekannter Pattern im Kontext betrieblicher Prozesse da. Alle gefundenen Pattern können über die Website http://www.bpmpatterns.org abgerufen werden. Im Rahmen der systematischen Literaturübersicht hat sich herausgestellt, dass Compliance-Pattern hinsichtlich des eingesetzten Kontexts die größten Ähnlichkeiten zu den im Rahmen dieser Arbeit gesuchten Pattern aufweisen. Die Überprüfung der Einhaltung eines Wartungsplans kann als Einhaltung der Compliance interpretiert werden, wenn beispielsweise eine bestimmte Aktivitätsreihenfolge eingehalten werden muss [vgl. ETHP14]. Bevor die einzelnen Pattern beschrieben werden, wird nachfolgend der Kontext, in dem die Pattern eingesetzt werden, genauer dokumentiert.

5.3 Pattern-Kontext

Wie in Kapitel 3.5 beschrieben, werden zur Erkennung von Fehlern bei der IT-Service-Wartung die generierten Ereignisse $x_1, x_2, ..., x_n$ der IT-Service-Komponenten sowie die Ereignisse über den Zustand einer IT-Service-Komponente durch die Prüfungseinheit *PE* auf Anti-Pattern überprüft. Eine solche Sequenz nacheinander eintreffender Ereignisse für einen IT-Service wird als Ereignis-Strom bezeichnet und ist in Anlehnung an [LSAB11] wie folgt definiert.

Definition 5.3: Ereignis-Strom

Ein Ereignis-Strom ist eine Sequenz zeitlich geordnet stattfindender Ereignisse $ES =< e_1, e_2, e_3, \dots, e_n >$ eines bestimmten IT-Services.

■

Ereignisse werden nachfolgend unterschieden in Vorgang-Ereignisse und Zustand-Ereignisse. Ein Vorgang-Ereignis ist ein von einer IT-Service-Komponente generiertes Ereignis. Ein Zustand-Ereignis ist ein Ereignis, das den Zustand einer IT-Service-Komponente beschreibt und durch eine Überwachungsanwendung generiert wird. Formal wird ein Vorgang-Ereignis wie folgt definiert.

Definition 5.4: Vorgang-Ereignis

Ein Vorgang-Ereignis ist ein Tupel $ve = (timestamp, st, app, op, prop, value)$ mit:

(i) $timestamp$ ist ein Zeitstempel, der angibt, wann das Ereignis ausgelöst wurde.
(ii) st ist die Service Template-*id,* die den IT-Service eindeutig identifiziert zu dem die IT-Service-Komponente gehört.
(iii) app ist die Node Template-*id,* durch die die IT-Service-Komponente identifiziert wird, die das Ereignis ausgelöst hat.
(iv) op referenziert die *Operation*, die auf der IT-Service-Komponente durchgeführt wurde und im entsprechenden Node Type, welches die IT-Service-Komponente beschreibt, hinterlegt ist.
(v) $prop$ beschreibt ein *Property* im Node Template, das bei der Operation angepasst wurde. Wenn op eine Operation ist, die keinen Parameter verändert, z.B. beim Starten einer IT-Service-Komponente, ist $prop$ leer.
(vi) $value$ ist der Wert, der bei der Operation $prop$ zugeordnet wird. Wenn $prop$ leer ist, dann ist $value$ ebenfalls leer.

Die Menge aller Vorgang-Ereignisse der zu überwachenden IT-Services wird als VEM definiert.

■

Ein Zustand-Ereignis wird wie folgt definiert.

Definition 5.5: Zustand-Ereignis

Ein Zustand-Ereignis ist ein Tupel $ze = (timestamp, app, state)$ mit:

(i) $timestamp$ ist ein Zeitstempel, der angibt, wann das Ereignis ausgelöst wurde.
(ii) app ist die Node Template-*id* durch die die IT-Service-Komponente identifiziert wird, dessen Zustand gemessen wurde.
(iii) $state$ beschreibt den Zustand der IT-Service-Komponente. Es sind nur Werte erlaubt, die im Node Type der IT-Service-Komponente beschrieben sind (*InstanceStates*).

Die Menge aller Zustand-Ereignisse der zu überwachenden IT-Services wird als ZEM definiert.

■

Vorgang-Ereignisse werden durch eine IT-Service-Komponente erzeugt und repräsentieren somit eine real durchgeführte Tätigkeit während einer IT-Service-Wartung. Eine im Wartungsplan geplante Tätigkeit wird durch eine sogenannte Aktivität repräsentiert. Eine Aktivität beschreibt eine *Operation*, die auf einem Node Template ausgeführt werden soll, z.B. starten oder stoppen. Das jeweilige Node Template kann durch die *id* eindeutig identifiziert werden[1]. Bei Konfigurationsänderungen ist diese Bestimmung jedoch nicht eindeutig, da beispielsweise ein anderer Konfigurationsparameter angepasst werden kann als der im Wartungsplan angegebene. Daher sind für die eindeutige Bestimmung von Konfigurationsaktivitäten zusätzlich zur Node Template *id* und der *Operation* die geänderten *Properties* zu berücksichtigen. Außerdem enthalten Aktivitäten einen Start- und Endzeitpunkt, die den Attributen *Start* und *Ende* der jeweiligen Transition entnommen werden. Start- und Endzeitpunkt geben an, in welchem Zeitraum die Aktivität ausgeführt werden muss. Zusätzlich enthält jede Aktivität die eindeutige *id* des Service Templates. Formal wird eine Aktivität wie folgt definiert:

Definition 5.6: Aktivität

Eine Aktivität ist ein Tupel $a = (st, app, op, prop, value, start, ende)$ mit:

(i) st ist die Service Template-*id,* die den IT-Service identifiziert.

[1] Durch das Filterschemata der aus der Transition ausgehenden Kante kann die durchzuführende Operation auch nur auf bestimmte Node Templates aus dem Nachbereich der Transition eingeschränkt sein (siehe Beschreibung der Funktionsweise von Filterschemata in Kapitel 4.4).

(ii) app ist die id des Node Templates auf dem die Operation durchgeführt wird.
(iii) op ist die *Operation*, die auf dem Node Template ausgeführt wird und im entsprechenden Node Type definiert ist.
(iv) $prop$ beschreibt ein *Property* im Node Template, das bei der Operation angepasst wird. Wenn op eine Operation ist, die keinen Parameter verändert, z.B. beim Starten einer IT-Service-Komponente, ist $prop$ leer.
(v) $value$ ist der Wert, der bei der Operation $prop$ zugeordnet wird. Wenn $prop$ leer ist, dann ist $value$ ebenfalls leer.
(vi) $start$ gibt an, wann die Aktivität frühestens stattfinden darf.
(vii) $ende$ gibt an, bis wann die Aktivität abgeschlossen sein muss.

Sei WPI die Menge aller Wartungspläne. Die Menge aller Aktivitäten eines Wartungsplans wird als AM_i mit $i \in WPI$ definiert. Die Menge aller Transitionen eines Wartungsplans wird als TM_i mit $i \in WPI$ definiert. Die Funktion $F: AM_i \rightarrow TM_i$ ordnet jeder Aktivität eine Transition zu.

■

Tabelle 5.1 zeigt wie die Attribute einer Aktivität aus einem Wartungsplan ermittelt werden können.

Tabelle 5.1: Mapping eines Wartungsplans zu Aktivitäten

Wartungsplan	**Aktivität**
ID des Service Templates, welches durch eine Markierung im Wartungsplan repräsentiert wird.	st
ID der Node Templates, die gegenüber dem Node Type gültig sind. Durch das Filterschema der ausgehenden Kante der Transition kann die Menge gültiger Node Templates zusätzlich eingeschränkt werden.	app
Operation, die durch die Transition repräsentiert wird.	op
Zu manipulierende Properties eines Node Templates, die im Filterschema der aus der Transition ausgehenden Kante deklariert sind.	$prop$
Der Wert, der einem Property im Filterschema zugeordnet wird.	$value$
Das Attribut Start, das die Transition enthält.	$start$
Das Attribut Ende, das die Transition enthält.	$ende$

Abbildung 5.1 zeigt einen Ausschnitt des Wartungsplans aus Abbildung 4.16 mit dem dargestellten Node Type und welche Aktivitäten sich daraus ergeben. Da im Filterschema eine

Einschränkung auf die Applikation MYAPPLICATION vorgenommen wird und insgesamt drei Parameter geändert werden, ergeben sich in dem Beispiel drei Aktivitäten. Angenommen, die Einschränkung auf MYAPPLICATION würde nicht bestehen und der Vorbereich der Transition CONFIGURE wäre mit drei Service Templates markiert, die jeweils gegenüber dem Node Type APPLICATION exakt ein gültiges Node Template beinhalten, so würden sich daraus insgesamt neun Aktivitäten ergeben. Wenn eine Transition schaltet können somit mehrere Aktivitäten ausgeführt werden.

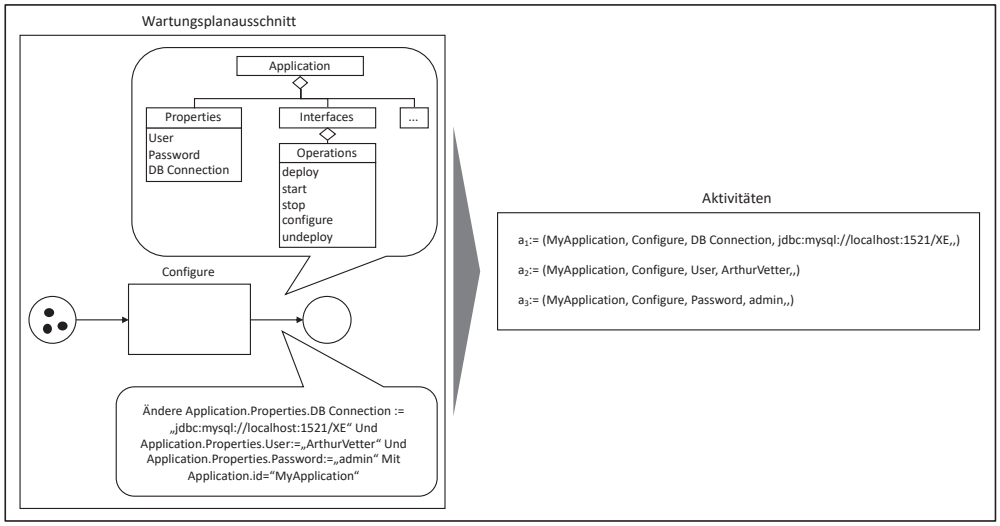


Abbildung 5.1: Ableitung von Aktivitäten aus einer Transition und einem Filterschema

Das Schalten einer Transition kann einer Bedingung unterliegen. Eine Bedingung kann beispielsweise sein, dass eine IT-Service-Komponente in einem bestimmten Zustand sein muss, bevor eine Aktivität ausgeführt werden kann. Die IT-Service-Komponente sollte durch eine vorherige Aktivität in diesen Zustand gebracht werden. Allerdings kann es vorkommen, dass beispielsweise durch einen Hardwaredefekt sich die IT-Service-Komponente nicht mehr im jeweiligen Zielzustand befindet. Ein solcher Ausfall ist zwar kein Wartungsfehler, hat jedoch trotzdem Auswirkungen auf die korrekte Durchführung eines IT-Services. Diese Art von Bedingungen, im folgenden Zustand-Bedingungen genannt, werden daher ebenfalls mittels Anti-Pattern überprüft. Allerdings wird dabei nicht die Entstehungsursache angegeben, warum die Zustand-Bedingung nicht erfüllt werden kann, sondern nur, dass die Zustand-Bedingung nicht eingehalten wird. Zustand-Bedingungen werden in einem Wartungsplan in Form von Transitionsinschriften modelliert.

Definition 5.7: Zustand-Bedingung

Eine Zustand-Bedingung ist ein Tupel $bed = (app, op, prop, zapp, state)$ mit:

(i) app ist die *id* des Node Templates auf dem die Operation durchgeführt wird.
(ii) op ist die *Operation*, die auf dem Node Template ausgeführt wird und im entsprechenden Node Type definiert ist.
(iii) $prop$ beschreibt ein *Property* im Node Template, das bei der Operation angepasst wird. Wenn op eine Operation ist, die keinen Parameter verändert, z.B. beim Starten einer IT-Service-Komponente, ist $prop$ leer.
(iv) $zapp$ ist die *id* des Node Templates, das sich in einem bestimmten Zustand befinden muss, damit die Operation ausgeführt werden kann.
(v) $state$ beschreibt den Zustand in dem sich $zapp$ befinden soll.

Sei WPI die Menge aller Wartungspläne. Die Menge aller Zustand-Bedingungen eines Wartungsplans wird als BM_i mit $i \in WPI$ definiert. Die Menge aller Transitionsinschriften eines Wartungsplans wird als TIM_i mit $i \in WPI$ definiert. Die Funktion $B: BM_i \rightarrow TIM_i$ ordnet jeder Zustand-Bedingung eine Transitionsinschrift zu.

■

Die Zuordnung der Elemente eines Wartungsplans zu einer Zustand-Bedingung kann Tabelle 5.2 entnommen werden.

Tabelle 5.2: Mapping eines Wartungsplans zu Zustand-Bedingungen

Wartungsplan	**Zustand-Bedingung**
ID der Node Templates, die gegenüber dem Node Type gültig sind. Durch das Filterschema der ausgehenden Kante der Transition kann die Menge gültiger Node Templates zusätzlich eingeschränkt werden.	app
Operation, die durch die Transition repräsentiert wird.	op
Zu manipulierende Properties eines Node Templates, die im Filterschema der aus der Transition ausgehenden Kante deklariert sind.	$prop$
Die ID der Node Templates, bei denen in der Transitionsinschrift das Element *InstanceState.State* (siehe Erweiterung TOSCA in Kapitel 4.4) geprüft wird. Durch das Filterschema der eingehenden Kante der Transition kann die Menge gültiger Node Templates eingeschränkt werden.	$Zapp$
Der Wert, der in der Transitionsinschrift dem Element *InstanceState.State* zugewiesen wird.	$state$

Einige Pattern benötigen zur Fehlererkennung Ereignisse, die in der Vergangenheit liegen. Daher ist eine Funktion notwendig, die Ereignisse für das zu überwachende Wartungsfenster historisiert. Hierfür wird mit dem relationalen Operator σ eine Selektion auf die Menge aller Vorgang-Ereignisse durchgeführt. Im Selektionsprädikat wird eine Einschränkung der auszuwählenden Vorgang-Ereignisse auf den Beginn und das Ende des zu untersuchenden Wartungsfensters vorgenommen. Eine solche Ereignis-Historie für Vorgang-Ereignisse wird wie folgt definiert.

Definition 5.8: Vorgang-Ereignis-Historie

Eine Vorgang-Ereignis-Historie

$$VEH := \sigma_{timestamp \geq Wartungsfensterbeginn \wedge timestamp \leq Wartungsfensterende}(VEM)$$

ist eine Selektion auf die Menge aller Vorgang-Ereignisse, die innerhalb des zu untersuchenden Wartungsfensters auftreten. Der Start eines Wartungsfensters wird als Wartungsfensterbeginn und das Ende des Wartungsfensters als Wartungsfensterende bezeichnet.

■

Eine Ereignis-Historie für Zustand-Ereignisse enthält im Gegensatz zu einer Vorgang-Ereignis-Historie nur das letzte Zustand-Ereignis eines bestimmten Node Templates. Um die

Zustand-Ereignis-Historie zu bilden, wird mit dem relationalen Operator Υ eine Gruppierung und Aggregation auf den größten Zeitstempel (*timestamp*) je Node Template (*app*) aller Zustand-Ereignisse (ZEM) gebildet. Um anschließend den Status (*state*) zum aggregierten Zeitstempel je Node Template zu erhalten, ist mithilfe des Operators $\bowtie$ ein Verbund mit der Menge aller Zustand-Ereignisse (ZEM) notwendig. Da dazu eine Kopie der Menge aller Zustand-Ereignisse benötigt wird, werden die beiden Mengen vor dem Verbund mithilfe des Operators ρ umbenannt.

Formal wird die Zustand-Ereignis-Historie wie folgt definiert.

Definition 5.9: Zustand-Ereignis-Historie

Eine Zustand-Ereignis-Historie

$$ZEH := \rho_x(ZEM) \bowtie_{\substack{x.app=y.app \wedge \\ x.timestamp=y.timestamp}} \rho_y\left(\gamma_{\text{app;max}(timestamp)}(ZEM)\right)$$

ist eine Selektion auf das letzte Zustand-Ereignis jedes Node Templates.

■

Außerdem werden zur Beschreibung der Pattern die zusätzlichen Funktionen Zeit, ZähleVE und ZähleA benötigt, die wie folgt definiert sind.

Definition 5.10: Zeit

Zeit ist eine Variable, die den aktuellen Zeitstempel enthält.

■

Definition 5.11: ZähleVE

ZähleVE ist eine Funktion $ZähleVE(ve, VEH)$, die die Anzahl des Vorgang-Ereignisses ve in der Vorgang-Ereignis-Historie VEH ausgibt.

■

Definition 5.12: ZähleA

ZähleA ist eine Funktion $ZähleA(a, M)$, die das Auftreten der Aktivität a in einer Menge M zählt und ausgibt.

■

Im Gegensatz zur Überprüfung von Compliance in betrieblichen Prozessen müssen bei der Überwachung von IT-Service-Wartungen nicht viele verschiedene Instanzen eines Ablaufs gleichzeitig überprüft werden. Im Rahmen von IT-Service-Wartungen werden Wartungen an einem IT-Service in der Regel sequentiell durchgeführt, um Seiteneffekte zu vermeiden. Dadurch entfällt die notwendige Zuordnung von überwachten Ereignissen zu mehreren Instanzen eines Prozesses wie beispielsweise in [ETHP14] oder beim klassischen Process Mining [AAMA11].

5.4 Pattern und Anti-Pattern für die IT-Service-Wartung

Damit die Ursache eines Fehlers online während der Wartungsdurchführung ermittelt werden kann müssen die Pattern bzw. Anti-Pattern so aussagekräftig sein, dass beim Erkennen eines Anti-Patterns, umgehend auf den Durchführungsfehler geschlossen werden kann. Dadurch ist keine weitere Analyse der Entstehungsursache notwendig, wie es beispielsweise in verwandten Arbeiten (siehe [XZWB14]) der Fall ist. Zudem sollen nicht nur der korrekte Ablauf, sondern auch Zustand-Bedingungen überprüft werden können, beispielsweise wenn eine Transition nur schalten darf, wenn sich eine bestimmte IT-Service-Komponente in einem bestimmten Zustand befindet. Außerdem sind geänderte Konfigurationsdaten zu überprüfen, um Konfigurationsfehler erkennen zu können. Um den korrekten Ablauf einer IT-Service-Wartung zu überprüfen, müssen die elementaren Ablaufmuster in einem Wartungsplan (siehe Kapitel 4.2.2) während der Durchführung überprüft werden.

Zur Beschreibung der einzelnen Pattern bzw. Anti-Pattern wird folgende Vorlage verwendet:

- Name
  Der Name des Patterns muss eindeutig sein und den Zweck des Patterns spiegeln.
- Beschreibung
  In der Beschreibung wird der Zweck des Patterns, d.h. der erwünschten Form, die im Rahmen der IT-Service Wartung auftreten soll, dokumentiert.
- Beispiel
  Ein Beispiel für das Pattern. Wenn nicht anders angegeben, wird zur besseren Verständlichkeit ein Beispiel auf Basis des in Abbildung 4.16 beschriebenen Wartungsplans verwendet.
- Instanzen
  Hier wird beschrieben, wie alle Instanzen des Patterns aus dem Wartungsplan ermittelt werden können.

- Fehler
  Hier wird beschrieben, welche Fehler bei der Wartungsdurchführung mit dem Anti-Pattern erkannt werden können.
- Anti-Pattern
  Das Anti-Pattern wird aus dem Pattern abgeleitet, und beschreibt, welche Bedingungen während der Durchführung der IT-Service-Wartung verletzt werden müssen, damit durch die Prüfungseinheit *PE* der Fehler erkannt wird. Das Anti-Pattern wird allgemein beschrieben, ist technologieunabhängig und beschreibt somit eine unerwünschte Form während der IT-Service-Wartung.
- Verwandte Anti-Pattern
  Unter Verwandte Anti-Pattern werden Anti-Pattern beschrieben mit denen teilweise ebenfalls dieselben Fehler erkannt werden können.

Nachfolgend werden auf Basis dieser Vorlage Pattern bzw. Anti-Pattern beschrieben, mit denen die vorgestellten Fehler erkannt werden können. Tabelle 5.3 zeigt eine Übersicht über alle Pattern bzw. Anti-Pattern. Die Pattern ANWESENHEIT, ABWESENHEIT, NACHFOLGER, DIREKTER NACHFOLGER, VORGÄNGER und DIREKTER VORGÄNGER basieren auf den beschriebenen Anti-Pattern zur Überprüfung von Compliance und Geschäftsprozessen in [ETHP14] und [DwAC98]. Sie werden jedoch auf den in Kapitel 5.3 beschriebenen Kontext angepasst. Die in dieser Arbeit vorgestellten Pattern und Anti-Pattern stellen keinen Anspruch auf Vollständigkeit, sondern ermöglichen die Entdeckung der in Kapitel 3.3 vorgestellten Wartungsfehler und decken die elementaren Ablaufmuster ab. Weitere Pattern, insbesondere zur Überprüfung komplexer Ablaufmuster, existieren, sind jedoch für die Entdeckung der vorgestellten Fehlersituationen nicht notwendig. Für eine Übersicht bereits bekannter Pattern zur Abbildung komplexer Ablaufmuster sei auf [FKLS17] verwiesen.

Tabelle 5.3: Pattern/Anti-Pattern-Übersicht

Kategorie	Pattern/ Anti-Pattern	Kurzbeschreibung
Auftreten	ANWESENHEIT	Beschreibt eine Aktivität, die durchgeführt werden muss.
	ABWESENHEIT	Beschreibt eine Aktivität, die nicht durchgeführt werden darf.
	ALTERNATIVE ANWESENHEIT	Beschreibt zwei Aktivitäten, von denen nur eine durchgeführt werden darf.
	ALTERNATIVE ABWESENHEIT	Beschreibt zwei Aktivitäten, von denen eine nicht auftreten darf, wenn die zweite Aktivität bereits durchgeführt wurde.
Reihenfolge	NACHFOLGER	Beschreibt welche Aktivitäten nach einer Aktivität auftreten müssen.
	DIREKTER NACHFOLGER	Beschreibt welche Aktivität direkt nach einer anderen Aktivität auftreten muss.
	VORGÄNGER	Beschreibt welche Aktivitäten vor einer Aktivität aufgetreten sein müssen.
	DIREKTER VORGÄNGER	Beschreibt welche Aktivität direkt vor einer Aktivität aufgetreten sein muss.
Vergleich	WERT	Beschreibt welcher Wert einem Parameter zugeordnet werden muss.
	ZUSTAND	Beschreibt in welchem Zustand sich eine IT-Service-Komponente befinden muss, bevor eine Operation auf dieser ausgeführt wird.

5.4.1 Pattern 1: NACHFOLGER

Beschreibung: Dieses Pattern beschreibt eine Aktivitäten-Folge, die angibt, welche Aktivität nach einer anderen Aktivität, jedoch vor einer dritten Aktivität stattfinden muss. Das Pattern kann nur für Aktivitäten verwendet werden, die nebenläufig zu anderen Aktivitäten durchgeführt werden, da bei diesem Pattern weitere zwischen der Aktivitäten-Folge durchgeführte Aktivitäten ignoriert werden. Mit diesem Pattern ist es möglich nebenläufige Aktivitäten nach einem AND-Split bzw. vor einem AND-Join in einem Wartungsplan zu überprüfen. Abbildung 5.2 veranschaulicht die Verwendung des Patterns NACHFOLGER.

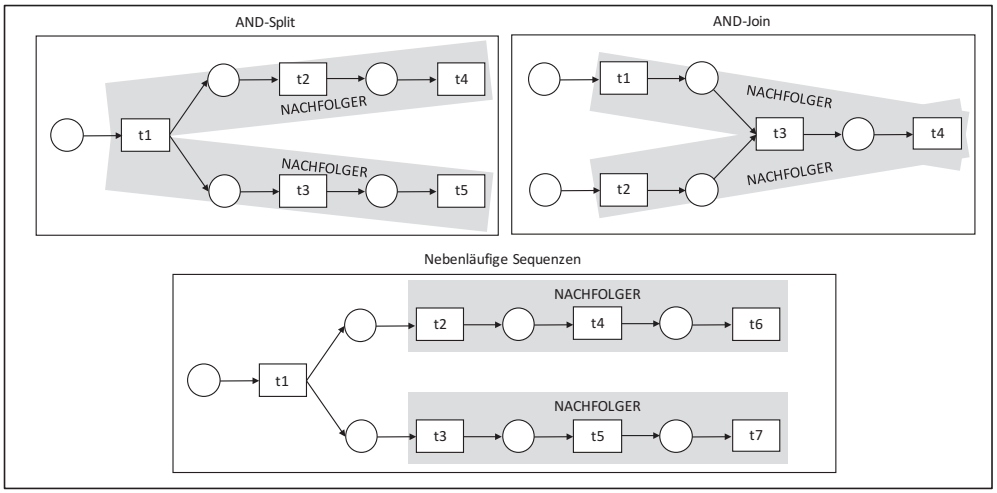


Abbildung 5.2: NACHFOLGER

Beispiel: In Abbildung 5.3 müssen nach der Durchführung von Aktivität A die Aktivitäten B und D folgen. Die Aktivität B muss vor der Aktivität C und die Aktivität D vor der Aktivität E durchgeführt werden. Dies bedeutet nicht, dass die Aktivitäten B und D direkt nach A durchgeführt werden müssen, sondern nur, dass diese generell nach A durchgeführt werden. Nach der Aktivität A können beispielsweise zunächst die Aktivitäten B und C durchgeführt werden, bevor die Aktivität D durchgeführt wird.

Instanzen: Um alle Instanzen des Patterns für einen Wartungsplan i zu ermitteln, wird eine Relation $P1_i \coloneqq AM_i \times AM_i \times AM_i \coloneqq \{(a_{akt}, a_{nach}, a_{entf}) | a_{akt}, a_{nach}, a_{entf} \in AM_i\}$ gebildet für die gilt,

- dass die in der Funktion F zugeordneten Transitionen für t_{akt} und t_{nach} eine gemeinsame Stelle im Nach- bzw. Vorbereich haben,
- t_{akt}, t_{nach} und t_{entf} im selben Pfad auftreten müssen,
- t_{akt} muss vor t_{entf} durchgeführt werden,
- t_{entf} und t_{akt} nicht durch eine gemeinsame Stelle verbunden sind.

Fehler: Mit dem Anti-Pattern kann der Fehler „Aktivität in der falschen Reihenfolge ausgeführt“ in nebenläufigen Aktivitäten erkannt werden. Außerdem können Konfigurationsfehler der Art „Syntaktischer Fehler“ erkannt werden, wenn ein Parameter in der falschen Reihenfolge geändert wird. Für die Bestimmung der richtigen Reihenfolge wird der Zeitpunkt des Auftretens eines Vorgang-Ereignisses berücksichtigt.

Anti-Pattern: Eine Fehlermeldung wird erzeugt, wenn folgende Bedingung bei der Auswertung eines gerade eintreffenden Vorgang-Ereignisses ve_{akt} im Ereignis-Strom ES zutrifft:

- eines der nachfolgenden Ereignisse ve_{nach} entspricht der Aktivität a_{entf}, jedoch wurde zwischen den beiden Ereignissen ve_{akt} und ve_{nach} die Aktivität a_{nach} nicht durchgeführt[2]:

$$\pi_{app,op,prop,value} ve_{akt} \in \pi_{a_{akt}.app,a_{akt}.op,a_{akt}.prop,a_{akt}.value} P1_i \succ \pi_{app,op,prop,value} ve_{nach} \in$$

$$\pi_{app,op,prop,value}\left(\pi_{a_{entf}}\left(\sigma_{\substack{a_{akt}.app=ve_{akt}.app \wedge a_{akt}.op=ve_{akt}.op \wedge \\ a_{akt}.prop=ve_{akt}.prop \wedge a_{akt}.value=ve_{akt}.value}} P1_i\right)\right) \wedge$$

$$\pi_{app,op,prop,value}\left(\pi_{a_{nach}}\left(\sigma_{\substack{a_{akt}.app=ve_{akt}.app \wedge a_{akt}.op=ve_{akt}.op \wedge \\ a_{akt}.prop=ve_{akt}.prop \wedge a_{akt}.value=ve_{akt}.value \wedge \\ a_{entf}.app=ve_{nach}.app \wedge a_{entf}.op=ve_{nach}.op \wedge \\ a_{entf}.prop=ve_{nach}.prop \wedge a_{entf}.value=ve_{nach}.value}} P1_i\right)\right) \notin$$

$$\pi_{app,op,prop,value}(\sigma_{ve_{akt}timestamp>timestamp \wedge ve_{nach}.timestamp<timestamp} VEH)$$

Als Fehlermeldung wird folgender Text ausgegeben:

Die Aktivität(en)

$$\pi_{app,op,prop,value}\left(\pi_{a_{nach}}\left(\sigma_{\substack{a_{akt}.app=ve_{akt}.app \wedge a_{akt}.op=ve_{akt}.op \wedge \\ a_{akt}.prop=ve_{akt}.prop \wedge a_{akt}.value=ve_{akt}.value \wedge \\ a_{entf}.app=ve_{nach}.app \wedge a_{entf}.op=ve_{nach}.op \wedge \\ a_{entf}.prop=ve_{nach}.prop \wedge a_{entf}.value=ve_{nach}.value}} P1_i\right)\right)$$

wurde(n) nicht nach der Aktivität $\pi_{app,op,prop,value} ve_{akt}$ durchgeführt.

Verwandte Anti-Pattern: Das Anti-Pattern DIREKTER NACHFOLGER ermöglicht ebenfalls die Erkennung des Fehlers „Aktivität in der falschen Reihenfolge ausgeführt“. Allerdings kann das Anti-Pattern DIREKTER NACHFOLGER Falschmeldungen für nebenläufige Aktivitäten liefern, da die eintreffenden Vorgang-Ereignisse in unterschiedlichen Reihenfolgen eintreffen

[2] Im Ereignisstrom nachfolgende Ereignisse werden mit dem Symbol $\succ$ dargestellt. Das Symbol π beschreibt eine Projektion, mit der Attribute extrahiert werden können. Die Menge der Attributnamen wird im Subskript definiert.

können, wie in Abbildung 5.3 gezeigt. Die Anwendung des Anti-Patterns DIREKTER NACHFOLGER eignet sich daher nur für den Einsatz bei nicht nebenläufigen Aktivitäten.

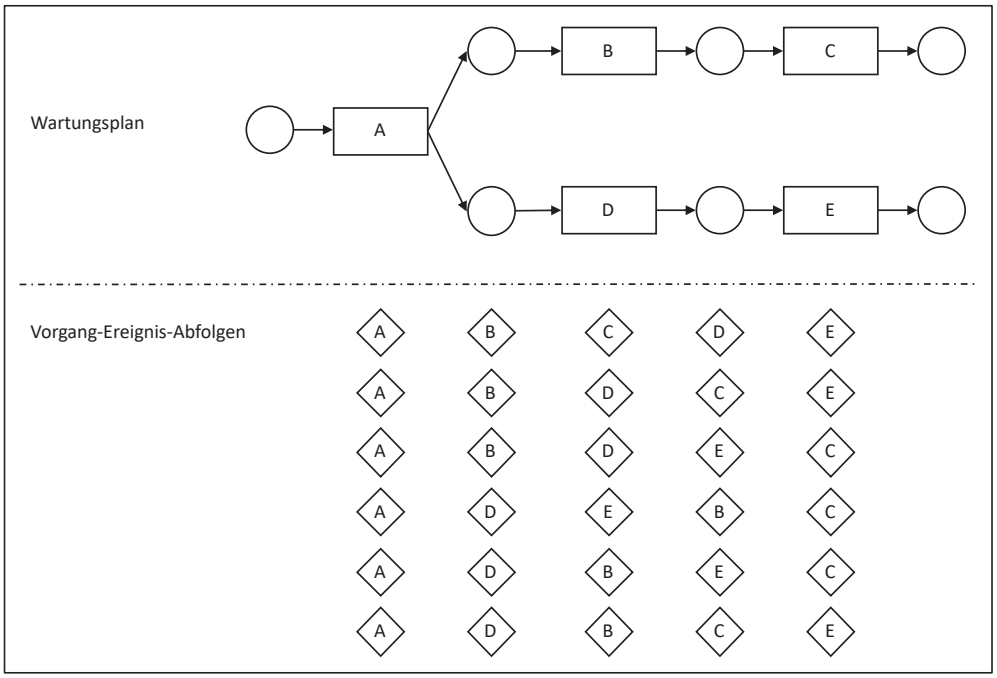


Abbildung 5.3: Beispiel Reihenfolge von Ereignissen

5.4.2 Pattern 2: DIREKTER NACHFOLGER

Beschreibung: Dieses Pattern beschreibt geordnete Aktivitäten-Paare, die angeben, welche Aktivität direkt nach einer anderen Aktivität stattfinden muss. Es dürfen bei diesem Pattern keine weiteren Aktivitäten zwischen der Vorgänger- und Nachfolger-Aktivität durchgeführt werden, d.h. es dürfen auch keine nebenläufigen Aktivitäten durchgeführt werden. Das Pattern kann eingesetzt werden, um XOR-Join, XOR-Split Ablaufmuster und nicht nebenläufige Sequenzen abzubilden. Abbildung 5.4 veranschaulicht die Anwendungsbereiche des Patterns DIREKTER NACHFOLGER.

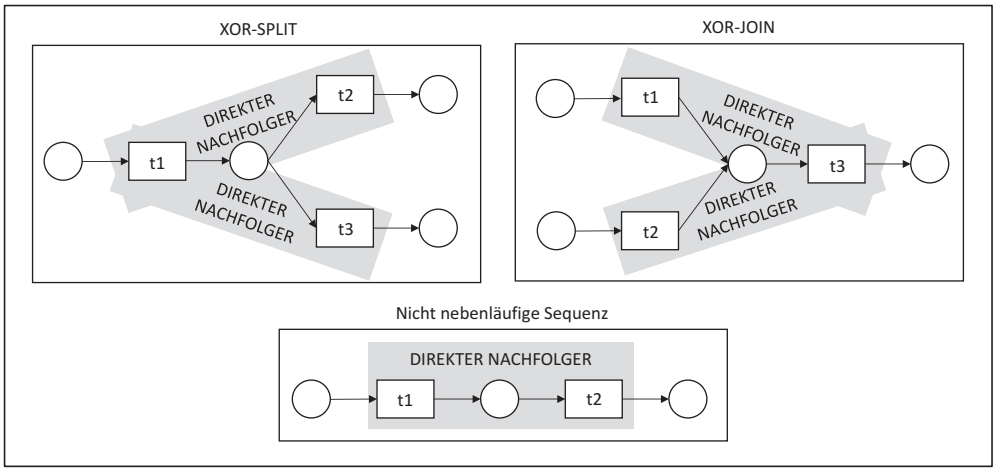


Abbildung 5.4: DIREKTER NACHFOLGER

Beispiel: Nach dem Stoppen von MYAPPLICATION, muss MYAPPLICATION direkt konfiguriert werden. Nach der Konfiguration von MYAPPLICATION muss MYAPPLICATION direkt wieder gestartet werden. Zwischen den Aktivitäten dürfen keine weiteren Aktivitäten durchgeführt werden, d.h. nach dem Stoppen von MYAPPLICATION darf MYAPPLICATION nicht sofort wieder gestartet werden.

Instanzen: Um alle Instanzen des Patterns für einen Wartungsplan i zu ermitteln, wird eine Relation $P2_i \coloneqq AM_i \times AM_i \coloneqq \{(a_{akt}, a_{nach}) \mid a_{akt}, a_{nach} \in AM_i\}$ gebildet für die gilt,

- dass die ihnen in der Funktion F zugeordneten Transitionen eine gemeinsame Stelle im Nach- bzw. Vorbereich haben, und
- dass die ihnen zugeordneten Transitionen nicht nebenläufig zu anderen Transitionen durchgeführt werden können, und

Fehler: Mit dem Anti-Pattern kann der Fehler „Aktivität in der falschen Reihenfolge ausgeführt“ bei nicht nebenläufigen Aktivitäten erkannt werden.

Anti-Pattern: Eine Fehlermeldung wird erzeugt, wenn folgende Bedingung bei der Auswertung eines gerade eintreffenden Vorgang-Ereignisses ve_{akt} im Ereignis-Strom ES zutrifft:

- das direkt nachfolgende Vorgang-Ereignis ve_{akt+1} desselben IT-Services entspricht nicht der erwarteten nachfolgenden Aktivität der aktuellen Aktivität:

$$\pi_{app,op,prop,value} ve_{akt} \in \pi_{a_{akt}.app,a_{akt}.op,a_{akt}.prop,a_{akt}.value} P2_i \succ \pi_{app,op,prop,value}(\sigma_{st=ve_{akt}.st} ve_{akt+1}) \notin \pi_{app,op,prop,value}\left(\pi_{a_{nach}}\left(\sigma_{\substack{a_{akt}.app=ve_{akt}.app \wedge a_{akt}.op=ve_{akt}.pp \wedge \\ a_{akt}.prop=ve_{akt}.prop \wedge a_{akt}.value=ve_{akt}.value}} P2_i\right)\right)$$

Als Fehlermeldung wird folgender Text ausgegeben:

Die Aktivität

$$\pi_{app,op,prop,value}\left(\pi_{a_{nach}}\left(\sigma_{\substack{a_{akt}.app=ve_{akt}.app \wedge a_{akt}.op=ve_{akt}.op \\ \wedge a_{akt}.prop=ve_{akt}.prop \wedge a_{akt}.value=ve_{akt}.value}} P2_i\right)\right)$$

wurde nicht direkt nach der Aktivität $\pi_{app,op,prop} ve_{akt}$ durchgeführt.

Verwandte Anti-Pattern: Das Anti-Pattern NACHFOLGER bietet ebenfalls die Funktionalität, nachfolgende Aktivitäten zu erkennen. Allerdings dürfen beim Anti-Pattern NACHFOLGER noch dritte Aktivitäten zwischen der gerade stattgefundenen Aktivität und der nachfolgenden Aktivität durchgeführt werden. Daher eignet sich der Einsatz des Anti-Patterns NACHFOLGER nicht für Aktivitäten-Paare, zwischen denen keine andere Aktivität durchgeführt werden darf.

5.4.3 Pattern 3: VORGÄNGER

Beschreibung: Dieses Pattern beschreibt geordnete Aktivitäten-Paare, die angeben, welche Aktivität vor einer anderen Aktivität durchgeführt werden muss. Das Pattern kann nur für Aktivitäten verwendet werden, die nebenläufig zu anderen Aktivitäten durchgeführt werden. Mit diesem Pattern ist es möglich, AND-Splits und AND-Joins sowie nebenläufige Sequenzen in einem Wartungsplan zu überprüfen. Abbildung 5.5 veranschaulicht die Verwendung des Patterns VORGÄNGER.

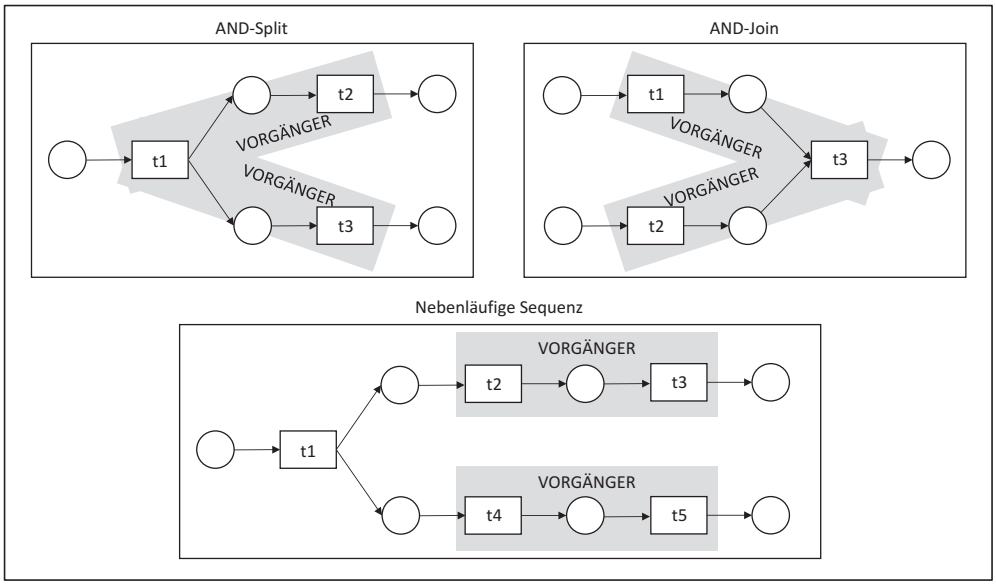


Abbildung 5.5: VORGÄNGER

Das Pattern VORGÄNGER ist symmetrisch zum Pattern NACHFOLGER. Warum trotzdem beide Pattern für die Fehlererkennung benötigt werden, wird im Bereich verwandte Pattern begründet.

Beispiel: In Abbildung 5.3 muss bei der Durchführung von Aktivität C, Aktivität B bereits durchgeführt worden sein. Bei der Durchführung von Aktivität B, muss Aktivität A bereits durchgeführt worden sein.

Instanzen: Um alle Instanzen des Patterns für einen Wartungsplan i zu ermitteln, wird eine Relation $P3_i \coloneqq AM_i \times AM_i \coloneqq \{(a_{akt}, a_{vor}) \mid a_{akt}, a_{vor} \in AM_i\}$ gebildet, für die gilt,

- dass die ihnen in der Funktion F zugeordneten Transitionen eine gemeinsame Stelle im Vor- bzw. Nachbereich haben, und
- dass die ihnen zugeordneten Transitionen nebenläufig zu anderen Transitionen durchgeführt werden können.

Fehler: Mit dem Anti-Pattern werden Fehler des Typs „Aktivität in der falschen Reihenfolge durchgeführt“ erkannt.

Anti-Pattern: Eine Fehlermeldung wird erzeugt, wenn folgende Bedingung bei der Auswertung eines gerade eintreffenden Vorgang-Ereignisses ve im Ereignis-Strom ES zutrifft:

- Ein Vorgang-Ereignis wird erkannt, dessen Vorgänger-Aktivität a_{vor} nicht in der Vorgang-Ereignis-Historie auftaucht[3]:

$$\pi_{app,op,prop,value} ve \in \pi_{a_{akt}.app,a_{akt}.op,a_{akt}.prop,a_{akt}.value} P3_i \wedge$$
$$\pi_{app,op,prop,value}\left(\pi_{a_{vor}}\left(\sigma_{\substack{a_{akt}.app=ve.app \wedge a_{akt}.op=ve.op \wedge \\ a_{akt}.prop=ve.prop \wedge a_{akt}.value=ve.value}} P3_i\right)\right) \notin$$
$$\pi_{app,op,prop,value} VEH$$

Als Fehlermeldung wird folgender Text ausgegeben:

Bevor die Aktivität $\pi_{app,op,prop,value} ve$ ausgeführt werden kann, muss/müssen die Aktivität(en) $\pi_{app,op,prop,vlaue}\left(\pi_{a_{Vor}}\left(\sigma_{\substack{a_{akt}.app=ve.app \wedge a_{akt}.op=ve.op \wedge \\ a_{akt}.prop=ve.prop \wedge a_{akt}.value=ve.value}} P3_i\right)\right)$ durchgeführt worden sein.

Verwandte Anti-Pattern: Mit dem Anti-Pattern DIREKTER VORGÄNGER kann in Sequenzen geprüft werden, ob die Vorgänger-Aktivität einer Aktivität bereits durchgeführt wurde. Analog zu den Anti-Pattern NACHFOLGER und DIREKTER NACHFOLGER wird beim Anti-Pattern DIREKTER VORGÄNGER jedoch nur die letzte Aktivität vor der aktuell erfassten überprüft. Beim Pattern NACHFOLGER wird zwar wie beim Pattern VORGÄNGER die korrekte Reihenfolge von zwei Aktivitäten beschrieben, jedoch unterscheidet sich ihre Überprüfung durch die Anti-Pattern zur Laufzeit. Bei einem Aktivitäten-Paar (a,b) kann ein Fehler durch das Anti-Pattern NACHFOLGER nur erkannt werden, wenn die Aktivität a durchgeführt wird und somit das entsprechende Vorgang-Ereignis erkannt wird. Wird jedoch die Aktivität a vergessen und nur die Aktivität b durchgeführt, ist nur das Anti-Pattern VORGÄNGER in der Lage diesen Fehler zu erkennen.

5.4.4 Pattern 4: DIREKTER VORGÄNGER

Beschreibung: Dieses Pattern beschreibt geordnete Aktivitäten-Paare, die angeben, welche Aktivität direkt vor einer anderen Aktivität durchgeführt werden muss. Im Gegensatz zum Pattern VORGÄNGER dürfen Aktivitäten dabei nicht nebenläufig zu anderen Aktivitäten durchgeführt werden. Das Pattern kann eingesetzt werden, um XOR-Split und XOR-Join

[3] Statt der Überprüfung der Vorgänger-Aktivität durch Auswertung von $P3_i$, könnte die Bedingung auch für die Anwendung mit $P1_i$ modifziert werden, da in den beiden Mengen die geordneten Paare lediglich anders geordnet sind, z.B. in $P1_i\ (a, b)$ und in $P3_i\ (b, a)$.

Ablaufmuster und nicht nebenläufige Sequenzen abzubilden. Abbildung 5.6 veranschaulicht die Anwendung des Patterns DIREKTER VORGÄNGER.

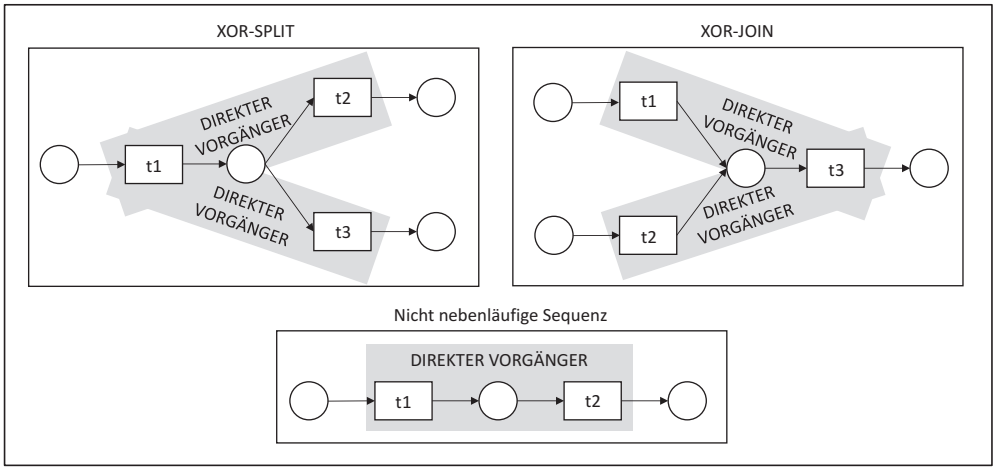


Abbildung 5.6: DIREKTER VORGÄNGER

Beispiel: Wenn MYAPPLICATION konfiguriert wird, dann muss zuvor MYAPPLICATION gestoppt worden sein.

Instanzen: Um alle Instanzen des Patterns für einen Wartungsplan i zu ermitteln, wird eine Relation $P4_i := AM_i \times AM_i := \{(a_{akt}, a_{vor}) \mid a_{akt}, a_{vor} \in AM_i\}$ gebildet für die gilt,

- dass die ihnen in der Funktion F zugeordneten Transitionen eine gemeinsame Stelle im Vor- bzw. Nachbereich haben, und
- dass die ihnen zugeordneten Transitionen nicht nebenläufig zu anderen Transitionen durchgeführt werden können.

Fehler: Mit dem Anti-Pattern werden Fehler des Typs „Aktivität in der falschen Reihenfolge durchgeführt“ erkannt.

Anti-Pattern: Eine Fehlermeldung wird erzeugt, wenn folgende Bedingung bei der Auswertung eines gerade eintreffenden Vorgang-Ereignisses ve im Ereignis-Strom ES zutrifft:

- Ein Vorgang-Ereignis wird erkannt, jedoch entspricht das letzte Vorgang-Ereignis in der Vorgang-Ereignis-Historie nicht der geplanten Vorgänger-Aktivität[4]:

[4] Wie beim Pattern VORGÄNGER könnte die Überprüfung der Vorgänger-Aktivität durch Auswertung von $P4_i$, stattdessen auch für die Auswertung mit $P2_i$ angepasst werden.

$$\pi_{app,op,prop,value} ve \in \pi_{a_{akt}.app,a_{akt}.op,a_{akt}.prop,a_{akt}.value} P4_i \wedge$$
$$\pi_{app,op,prop,value}\left(\pi_{a_{vor}}\left(\sigma_{\substack{a_{akt}.app=ve.app \wedge a_{akt}.op=ve.op \wedge \\ a_{akt}.prop=ve.prop \wedge a_{akt}.value=ve.value}} P4_i\right)\right) \notin$$
$$\pi_{app,op,prop,value}(\sigma_{\max(timestamp)}(\sigma_{timestamp<ve.timestamp \wedge st=ve.st} VEH))$$

Als Fehlermeldung wird folgender Text ausgegeben:

Die notwendige Vorgänger-Aktivität

$$\pi_{app,op,prop,value}\left(\pi_{a_{Vor}}\left(\sigma_{\substack{a_{akt}.app=ve.app \wedge a_{akt}.op=ve.pp \wedge \\ a_{akt}.prop=ve.prop \wedge a_{akt}.value=ve.value}} P4_i\right)\right)$$

wurde nicht direkt vor der Aktivität $\pi_{app,op,prop,value} ve$ durchgeführt.

Verwandte Anti-Pattern: Das Anti-Pattern VORGÄNGER bietet ebenfalls die Funktionalität, vorab durchzuführende Aktivitäten zu erkennen. Allerdings dürfen beim Anti-Pattern VORGÄNGER noch dritte Aktivitäten zwischen der gerade stattgefundenen Aktivität und der zuvor durchzuführenden Aktivität stattfinden. Daher eignet sich der Einsatz des Anti-Patterns VORGÄNGER nicht für nicht nebenläufige Aktivitäten.

5.4.5 Pattern 5: ANWESENHEIT

Beschreibung: Dieses Pattern beschreibt geplante Aktivitäten einer IT-Service Wartung, die innerhalb des Wartungsfensters zwingend durchgeführt werden müssen. Aktivitäten, bei denen bei der Entstehung des Wartungsplans unklar ist, ob diese zur Laufzeit tatsächlich durchgeführt werden, sind nicht Teil dieses Patterns. Solche Aktivitäten können bei oder nach einem XOR-Split auftreten. In diesem Fall entscheidet sich erst zur Laufzeit, welche weiteren Aktivitäten nach dem XOR-Split durchgeführt werden müssen. Abbildung 5.7 veranschaulicht die Anwendung des Patterns.

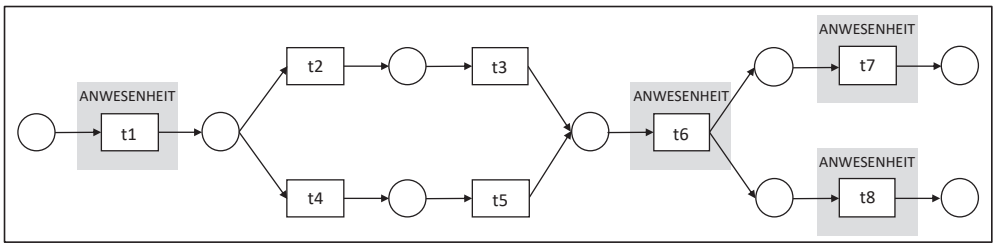


Abbildung 5.7: ANWESENHEIT

Beispiel: MYAPPLICATION muss während der Wartung einmal gestoppt und gestartet werden.

Instanzen: Um alle Instanzen des Patterns für einen Wartungsplan zu ermitteln, wird eine Menge $P5_i$ aller im Wartungsplan i geplanten Aktivitäten gebildet, für die gilt,

- dass die ihnen in der Funktion F zugeordneten Aktivitäten unabhängig eines Ablaufmusters immer durchgeführt werden müssen.

Fehler: Mit dem Anti-Pattern werden Fehler des Typs „Aktivität vergessen“ erkannt.

Anti-Pattern: Mit dem Anti-Pattern wird überprüft, ob eine der Pattern-Instanzen nicht ausgeführt wurde. Eine Fehlermeldung wird erzeugt, wenn folgende Bedingung zutrifft:

- Eine Aktivität wurde zu selten innerhalb des geplanten Zeitraums erzeugt:

$$\exists\, a \in \sigma_{Zeit>a.ende} P5_i \wedge$$
$$Zähle VE\left(\pi_{app,op,prop,value} a,\ \pi_{app,op,prop,value}\left(\sigma_{timestamp \geq a.start \wedge timestamp \leq a.ende} VEH\right)\right) <$$
$$Zähle A\left(\pi_{app,op,prop,value} a, \pi_{app,op,prop,value}\left(\sigma_{start \geq a.start \wedge ende \leq a.ende} P5_i\right)\right)$$

Als Fehlermeldung wird folgender Text ausgegeben:

Die Aktivität $\pi_{app,op,prop,value} a$ wurde zwischen $a.start$ und $a.ende$ nur $Zähle VE\left(\pi_{app,op,prop,value} a, \pi_{app,op,prop,value}\left(\sigma_{timestamp \geq a.start \wedge timestamp \leq a.ende} VEH\right)\right)$ mal, statt $Zähle A\left(\pi_{app,op,prop,value} a, \pi_{app,op,prop,value}\left(\sigma_{start \geq a.start \wedge ende \leq a.ende} P5_i\right)\right)$ mal durchgeführt.

Verwandte Anti-Pattern: Mit den Anti-Pattern NACHFOLGER, DIREKTER NACHFOLGER, VORGÄNGER und DIREKTER VORGÄNGER können teilweise ebenfalls vergessene Aktivitäten erkannt werden. Allerdings lassen sich mit diesen Pattern nicht alle vergessenen Aktivitäten erkennen, wenn gemäß Wartungsplan mindestens drei direkt aufeinanderfolgende Aktivitäten durchgeführt werden müssen. Abbildung 5.8 zeigt eine Situation, in der nur mit dem Anti-Pattern ANWESENHEIT die vergessene Aktivität C erkannt werden kann, wenn nur die Vorgang-Ereignisse A und E eintreten. Zudem lässt sich nur mit dem Anti-Pattern ANWESENHEIT erkennen, wenn eine IT-Service-Wartung generell vergessen wurde durchzuführen, d.h. keine einzige Aktivität wurde durchgeführt. Ein weiterer Unterschied ist, dass bei den Anti-Pattern NACHFOLGER, DIREKTER NACHFOLGER, VORGÄNGER und DIREKTER VORGÄNGER das Auftreten innerhalb des geplanten Zeitraums nicht geprüft wird, sondern nur das fehlende Auftreten einer Aktivität in Abhängigkeit einer anderen Aktivität.

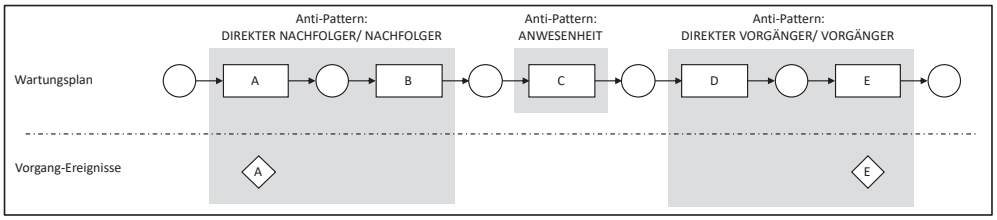


Abbildung 5.8: Beispiel Anti-Pattern ANWESENHEIT

5.4.6 Pattern 6: ALTERNATIVE ANWESENHEIT

Beschreibung: Dieses Pattern beschreibt zwei geplante Aktivitäten einer IT-Service Wartung, von denen nur eine innerhalb des Wartungsfensters durchgeführt werden darf. Solche alternativen Aktivitäten kommen nach dem Ablaufmuster XOR-Split vor.

Beispiel: In Abbildung 5.7 können entweder die Transitionen t2 und t3 oder t4 und t5 ausgeführt werden.

Instanzen: Um alle Instanzen des Patterns für einen Wartungsplan i zu ermitteln, wird eine Relation $P6_i := AM_i \times AM_i := \{(a_{akt}, a_{alt}) \mid a_{akt}, a_{alt} \in AM_i\}$ gebildet für die gilt,

- dass die ihnen zugeordneten Transitionen nicht gemeinsam in einem Pfad auftreten können.

In Abbildung 5.7 ergeben sich damit folgende Tupel: (t2,t4), (t2,t5), (t3,t4), (t3,t5), (t4,t2), (t4,t3), (t5,t2) und (t5,t3).

Fehler: Mit dem Anti-Pattern werden Fehler des Typs „Aktivität vergessen“ erkannt.

Anti-Pattern: Eine Fehlermeldung wird erzeugt, wenn folgende Bedingung zutrifft. Um Falschmeldungen zu vermeiden, wenn bereits mindestens zwei sich gegenseitig ausschließende Aktivitäten durchgeführt wurden, wird zusätzlich geprüft, welche alternative Aktivität zuerst durchgeführt wurde.

- Eine der alternativen Aktivitäten wurde zu selten innerhalb des geplanten Zeitraums durchgeführt:

$$\exists\, a \in \pi_{a_{akt}}(\sigma_{Zeit>a_{akt}.ende} P6_i) \wedge$$
$$ZähleVE\left(\begin{matrix}\pi_{app,op,prop,value} a,\\ \pi_{app,op,prop,value}\left(\sigma_{timestamp\geq a.start \,\wedge\, timestamp\leq a.ende} VEH\right)\end{matrix}\right) <$$
$$ZähleA\left(\begin{matrix}\pi_{app,op,prop,value} a, \pi_{a_{akt}.app,a_{akt}.op,a_{akt}.prop,a_{akt}.value}\\ \left(\sigma_{a_{akt}.start\geq a.start \,\wedge\, a_{akt}.ende\leq a.ende} P6_i\right)\end{matrix}\right) \wedge$$
$$\pi_{a_{alt}.app,a_{alt}.op,a_{alt}.prop,a_{alt}.value}$$
$$\left(\sigma_{\begin{matrix}a_{akt}.app=a.app\wedge a_{akt}.op=a.op\wedge a_{akt}.prop=a.prop\wedge\\ a_{akt}.value=a.value\wedge a_{akt}.start=a.start\wedge a_{akt}.ende=a.ende\end{matrix}} P6_i\right) \notin$$
$$\pi_{app,op,prop,value}(VEH \bowtie \gamma_{\min(timestamp)}($$
$$\left(\left(\rho_{P61}\left(\sigma_{\begin{matrix}a_{akt}.app=a.app\wedge a_{akt}.op=a.op\wedge a_{akt}.prop=a.prop\wedge\\ a_{akt}.value=a.value\wedge a_{akt}.start=a.start\wedge a_{akt}.ende=a.ende\end{matrix}} P6_i\right)\right)\right.$$
$$\left.\bowtie_{P61.a_{alt}=P6_i.a_{akt}} P6_i\right)$$
$$\bowtie_{\begin{matrix}(P6_i.a_{alt}.app=VEH.app\wedge P6_i.a_{alt}.op=VEH.op\wedge P6_i.a_{alt}.prop=VEH.prop\wedge P6_i.a_{alt}.value=\\ VEH.value\wedge P6_i.a_{alt}start\leq VEH.timestamp\wedge P6_i.a_{alt}ende\geq VEH.timestamp)\vee\\ (P61.a_{alt}.app=VEH.app\wedge P61.a_{alt}.op=VEH.op\wedge P61.a_{alt}.prop=VEH.prop\wedge P61.a_{alt}.value=\\ VEH.value\wedge P61.a_{alt}start\leq VEH.timestamp\wedge P61.a_{alt}ende\geq VEH.timestamp)\end{matrix}} VEH))$$

Als Fehlermeldung wird folgender Text ausgegeben:

Die Aktivität $\pi_{app,op,prop,value} a$ wurde zwischen $a.start$ und $a.ende$
$ZähleVE\left(\begin{matrix}\pi_{app,op,prop,value} a, \pi_{app,op,prop,value}\\ \left(\sigma_{timestamp\geq a.start \,\wedge\, timestamp\leq a.ende} VEH\right)\end{matrix}\right)$ mal, statt
$ZähleA\left(\begin{matrix}\pi_{app,op,prop,value} a, \pi_{a_{akt}.app,a_{akt}.op,a_{akt}.prop,a_{akt}.value}\\ \left(\sigma_{a_{akt}.start\geq a.start \,\wedge\, a_{akt}.ende\leq a.ende} P6_i\right)\end{matrix}\right)$ mal durchgeführt.

Verwandte Anti-Pattern: Mit dem Anti-Pattern ANWESENHEIT können ebenfalls vergessene Aktivitäten erkannt werden, jedoch werden damit nur Aktivitäten erkannt, die immer durchgeführt werden. Alternative Aktivitäten können mit dem Anti-Pattern ANWESENHEIT nicht erkannt werden, bzw. das Anti-Pattern würde Falschmeldungen verursachen.

5.4.7 Pattern 7: ABWESENHEIT

Beschreibung: Dieses Pattern beschreibt, welche Aktivitäten während der Wartung gemäß Wartungsplan nicht durchgeführt werden dürfen.

Beispiel: Gemäß Wartungsplan darf MYAPPLICATION während der IT-Service-Wartung nicht deployed oder undeployed werden. Außerdem dürfen die Properties *User* und *Password* nicht geändert werden. MYDATABASE darf während der IT-Service-Wartung nicht

deinstalliert, installiert, gestartet oder gestoppt werden. MYAPPLICATION muss jedoch während der Wartung exakt einmal gestoppt, konfiguriert und gestartet werden.

Instanzen: Instanzen dieses Patterns sind Aktivitäten, die während einer IT-Service-Wartung auftreten können, aber nicht im Wartungsplan enthalten sind. Alle möglichen Aktivitäten lassen sich durch die im Wartungsplan als Markierung enthaltenen Service Templates ermitteln. Aus diesen Service Templates können alle vorhandenen Node Templates sowie die auf ihnen durchführbaren Operationen und änderbaren Properties abgeleitet werden. Sei diese Menge aller möglichen Aktivitäten für eine IT-Service-Wartung j GAM_j, dann sind die Instanzen des Patterns für den dazugehörigen Wartungsplan i $P7_i \coloneqq GAM_j / AM_i$.

Fehler: Mit dem Anti-Pattern werden Fehler des Typs „unnötige Aktivität durchgeführt“ erkannt.

Anti-Pattern: Eine Fehlermeldung wird erzeugt, wenn eine der folgenden Bedingungen bei der Auswertung eines gerade eintreffenden Vorgang-Ereignisses ve im Ereignis-Strom ES für eine Pattern-Instanz zutrifft:

- Es existiert eine Aktivität, die dem Vorgang-Ereignis entspricht und innerhalb des geplanten Zeitraums erzeugt wurde, jedoch häufiger als erlaubt vorgekommen ist:

$$\exists\, a \in \sigma_{app=ve.app \wedge op=ve.op \wedge prop=ve.prop \wedge value=ve.value} AM_i \wedge \\ \wedge start \leq ve.timestamp \wedge ende \geq ve.timestamp$$

$$ZähleVE\left(\pi_{app,op,prop,value} ve,\ \pi_{app,op,prop,value}(\sigma_{timestamp \geq a.start \,\wedge\, timestamp \leq a.ende} VEH)\right) > \\ ZähleA\left(a, \left(\sigma_{ve.timestamp \geq start \,\wedge\, ve.timestamp \leq ende} AM_i\right)\right)$$

- Es existiert eine Aktivität, die dem Vorgang-Ereignis entspricht, jedoch wurde die Aktivität außerhalb des geplanten Zeitraums erzeugt:

$$\exists\, a \in (\sigma_{app=ve.app \wedge op=ve.op \wedge prop=ve.prop \wedge value=ve.value} AM_i) \wedge \\ a \notin (\sigma_{app=ve.app \wedge op=ve.op \wedge prop=ve.prop \wedge value=ve.value \wedge start \leq ve.timestamp \wedge ende \geq ve.timestamp} AM_i)$$

- Es wurde eine Aktivität durchgeführt, die nicht geplant war:

$$\pi_{app,op,prop} ve \notin \pi_{app,op,prop} AM_i.$$

Als Fehlermeldung wird entsprechend einer der folgenden Texte ausgegeben:

Die Aktivität $\pi_{app,op,prop,value} ve$ wurde zwischen $a.start$ und $a.ende$ $ZähleVE\left(\pi_{app,op,prop,value} ve,\ \pi_{app,op,prop,value}(\sigma_{timestamp \geq a.start \,\wedge\, timestamp \leq a.ende} VEH)\right)$ mal, statt maximal $ZähleA\left(a, \left(\sigma_{ve.timestamp \geq start \,\wedge\, ve.timestamp \leq ende} AM_i\right)\right)$ mal durchgeführt.

Die Aktivität $\pi_{app,op,prop,value}ve$ wurde außerhalb eines geplanten Zeitraums durchgeführt.

Die Aktivität $\pi_{app,op,prop}ve$ sollte nicht durchgeführt werden.

Verwandte Anti-Pattern: Mit den Anti-Pattern DIREKTER NACHFOLGER, NACHFOLGER, VORGÄNGER und DIREKTER VORGÄNGER können teilweise ebenfalls unnötig durchgeführte Aktivitäten erkannt werden, wenn die entsprechende nachfolgende bzw. vorangehende Aktivität nicht durchgeführt wurde. Allerdings lässt sich mit diesen Anti-Pattern nicht prüfen, ob es sich um eine generell nicht erlaubte Aktivität handelt. Zudem können nicht alle unnötigen Aktivitäten erkannt werden, wenn gemäß Wartungsplan mindestens drei direkt aufeinanderfolgende Aktivitäten durchgeführt werden müssen. Abbildung 5.9 zeigt eine Situation, in der nur mit dem Anti-Pattern ABWESENHEIT die nicht erlaubte Aktivität G erkannt werden kann. Die Aktivitäten F und H können auch mit den anderen, zum Anti-Pattern ABWESENHEIT verwandten, Anti-Pattern erkannt werden.

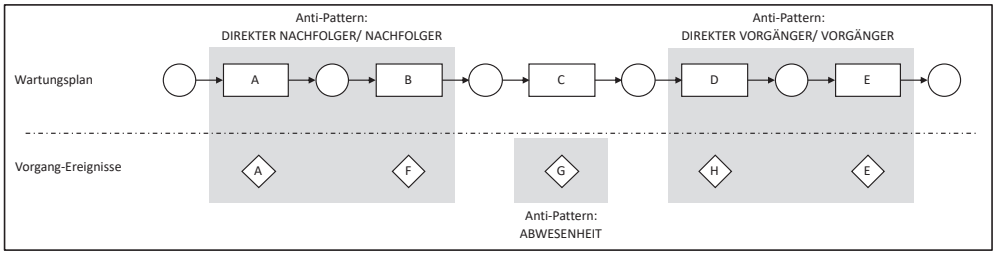


Abbildung 5.9: Beispiel Anti-Pattern ABWESENHEIT

Unnötige alternative Aktivitäten werden über das Pattern ALTERNATIVE ABWESENHEIT überprüft.

5.4.8 Pattern 8: ALTERNATIVE ABWESENHEIT

Beschreibung: Dieses Pattern beschreibt zwei alternative Aktivitäten, von denen eine Aktivität nicht auftreten darf, wenn die andere bereits aufgetreten ist. Solche Aktivitäten kommen nach dem Ablaufmuster XOR-Split vor.

Beispiel: In Abbildung 5.7 dürfen entweder die Transitionen t2 und t3 oder t4 und t5 ausgeführt werden, jedoch nicht t2, t3, t4 und t5. Ein anderes Beispiel ist, wenn eine Aktivität, z.B. Server starten, nach einem XOR-Split in jeweils beiden folgenden Pfaden auftritt. In diesem Fall darf die Aktivität, obwohl zweimal im Ablaufmodell vorhanden, nur maximal einmal auftreten.

Instanzen: Die Instanzen dieses Patterns für einen Wartungsplan i entsprechen denselben Instanzen des Patterns ALTERNATIVE ANWESENHEIT, d.h. $P8_i = P6_i$. Das Pattern wird trotzdem als eigenständiges Pattern beschrieben, da sich die Anti-Pattern der Pattern ALTERNATIVE ANWESENHEIT und ALTERNATIVE ABWESENHEIT unterscheiden.

Fehler: Mit dem Anti-Pattern werden Fehler des Typs „unnötige Aktivität durchgeführt" erkannt.

Anti-Pattern: Eine Fehlermeldung wird erzeugt, wenn folgende Bedingung bei der Auswertung eines gerade eintreffenden Vorgang-Ereignisses ve im Ereignis-Strom ES zutrifft. Um Falschmeldungen zu vermeiden, wenn bereits mindestens zwei sich gegenseitig ausschließende Aktivitäten durchgeführt wurden, wird zusätzlich geprüft, welche alternative Aktivität zuerst durchgeführt wurde.

- Es existiert eine Aktivität, die dem Vorgang-Ereignis entspricht und innerhalb des geplanten Zeitraums erzeugt wurde, jedoch wurde bereits zuvor eine alternative Aktivität ausgeführt:

$$\exists\, a \in \pi_{a_{akt}} \left(\sigma_{\substack{a_{akt}.app=ve.app \wedge a_{akt}.op=ve.op \wedge a_{akt}.prop=ve.prop \wedge a_{akt}.value=ve.value \wedge \\ a_{akt}.start \leq ve.timestamp \wedge a_{akt}.ende \geq ve.timestamp}} P8_i \right) \wedge$$

$$\pi_{a_{alt}.app, a_{alt}.op, a_{alt}.prop, a_{alt}.value}$$

$$\left(\sigma_{\substack{a_{akt}.app=a.app \wedge a_{akt}.op=a.op \wedge a_{akt}.prop=a.prop \wedge \\ a_{akt}.value=a.value \wedge a_{akt}.start=a.start \wedge a_{akt}.ende=a.ende}} P8_i \right) \in$$

$$\pi_{app,op,prop,value}(VEH \bowtie \gamma_{\min(timestamp)}((($$

$$\left(\rho_{P81} \left(\sigma_{\substack{a_{akt}.app=ve.app \wedge a_{akt}.op=ve.op \wedge a_{akt}.prop=ve.prop \wedge \\ a_{akt}.value=ve.value \wedge a_{akt}.start=a.start \wedge a_{akt}.ende=a.ende}} P8_i \right) \right)$$

$$\bowtie_{P81.a_{alt}=P8_i.a_{akt}} P8_i))$$

$$\bowtie (P8_i.a_{alt}.app=VEH.app \wedge P8_i.a_{alt}.op=VEH.op \wedge P8_i.a_{alt}.prop=VEH.prop \wedge P8_i.a_{alt}.value= VEH))$$

$$VEH.value \wedge P8_i.a_{alt}start \leq VEH.timestamp \wedge P8_i.a_{alt}ende \geq VEH.timestamp) \vee$$

$$(P81.a_{alt}.app=VEH.app \wedge P81.a_{alt}.op=VEH.op \wedge P81.a_{alt}.prop=VEH.prop \wedge P81.a_{alt}.value=$$

$$VEH.value \wedge P81.a_{alt}start \leq VEH.timestamp \wedge P81.a_{alt}ende \geq VEH.timestamp)$$

Als Fehlermeldung wird entsprechend folgender Text ausgegeben:

Die Aktivität $\pi_{app,op,prop,value} ve$ darf nicht ausgeführt werden, da bereits die Aktivität $\pi_{app,op,prop,value}(VEH \bowtie \gamma_{\min(timestamp)}((\left(\rho_{P81}\left(\sigma_{\substack{a_{akt}.app=ve.app \wedge a_{akt}.op=ve.op \wedge a_{akt}.prop=ve.prop \wedge \\ a_{akt}.value=ve.value \wedge a_{akt}.start=a.start \wedge a_{akt}.ende \geq a.ende}} P8_i\right)\right) \bowtie_{P81.a_{alt}=P8_i.a_{akt}} P8_i) \bowtie_{\substack{(P8_i.a_{alt}.app=VEH.app \wedge P8_i.a_{alt}.op=VEH.op \wedge P8_i.a_{alt}.prop=VEH.prop \wedge P8_i.a_{alt}.value= \\ VEH.value \wedge P8_i.a_{alt} start \leq VEH.timestamp \wedge P8_i.a_{alt} ende \geq VEH.timestamp) \vee \\ (P81.a_{alt}.app=VEH.app \wedge P81.a_{alt}.op=VEH.op \wedge P81.a_{alt}.prop=VEH.prop \wedge P81.a_{alt}.value= \\ VEH.value \wedge P81.a_{alt} start \leq VEH.timestamp \wedge P81.a_{alt} ende \geq VEH.timestamp)}} VEH))$ durchgeführt wurde.

Verwandte Anti-Pattern: Mit dem Anti-Pattern ABWESENHEIT lassen sich ebenfalls nicht erlaubte Aktivitäten erkennen. Jedoch wird beim Anti-Pattern ABWESENHEIT nicht geprüft, ob eine alternative Aktivität unnötigerweise durchgeführt wird, weil bereits eine konkurrierende alternative Aktivität ausgeführt wurde.

5.4.9 Pattern 9: WERT

Beschreibung: Dieses Pattern beschreibt den Wert eines zu ändernden Parameters. Wenn beispielsweise die IP-Adresse eines Servers geändert werden soll, so wäre in diesem Fall der Parameter beispielsweise „IP Server“ und der geplante Wert „172.16.9.98“.

Beispiel: Der eingetragene Wert der IT-Service-Komponente MYAPPLICATION für den Parameter DB CONNECTION muss JDBC:MYSQL://LOCALHOST:1521/XE sein.

Instanzen: Um alle Instanzen des Patterns für einen Wartungsplan zu ermitteln, wird eine Selektion auf alle im Wartungsplan geplanten Aktivitäten durchgeführt $P9_i \coloneqq \pi_{app,op,prop,value}(\sigma_{prop \neq NULL} AM_i)$, bei denen ein Property eines Node Templates geändert wird.

Fehler: Es können die Fehler „Aktivität falsch ausgeführt“, „lexikalischer Fehler“, „lokale Inkonsistenz“, „globale Inkonsistenz“ und „Typo“ erkannt werden.

Anti-Pattern: Eine Fehlermeldung wird erzeugt, wenn folgende Bedingung bei der Auswertung eines gerade eintreffenden Vorgang-Ereignisses ve im Ereignis-Strom ES zutrifft:

- Der $value$-Wert des Vorgang-Ereignisses entspricht nicht dem $value$-Wert der entsprechenden Aktivität:

$$\pi_{app,op,prop} ve \in \pi_{app,op,prop} P9_i \wedge$$
$$\pi_{app,op,prop,value} ve \notin P9_i$$

Als Fehlermeldung wird folgender Text ausgegeben:

Der geänderte Wert $\pi_{app,op,prop,value} ve$ entspricht nicht dem geplanten Wert $\pi_{app,op,prop,value}(\sigma_{app=ve.app,op=ve.opp,prop=ve.prop} P9_i)$.

Verwandte Anti-Pattern: Das Anti-Pattern kann als weitere Detaillierung des Anti-Patterns ANWESENHEIT interpretiert werden. Allerdings ist der Zweck des Anti-Patterns ANWESENHEIT nicht die Entdeckung von Fehlern bei geänderten Parameterwerten, sondern ob überhaupt die richtige Operation ausgeführt wird und der richtige Parameter geändert wird.

5.4.10 Pattern 10: ZUSTAND

Beschreibung: Dieses Pattern beschreibt, in welchem Zustand sich eine IT-Service-Komponente befinden muss, damit eine Aktivität ausgeführt werden darf.

Beispiel: Wenn MYAPPLICATION gestartet wird, muss der letzte Zustand von MYAPPLICATION STOPPED sein. Der letzte Zustand von MYDATABASE muss STARTED sein.

Instanzen: Instanzen dieses Patterns für einen Wartungsplan i sind alle Zustand-Bedingungen in der Menge BM_i.

Fehler: Das Anti-Pattern dient nicht zur Erkennung einer der in Kapitel 3.3 vorgestellten Fehler. Stattdessen kann mit dem Anti-Pattern geprüft werden, ob sich eine IT-Service-Komponente im „richtigen“ Zustand befindet, beispielsweise ob ein Server offline ist. Die Ursache dafür, dass sich eine IT-Service-Komponente nicht im „richtigen“ Zustand befindet, kann ein Hardwareausfall sein. Unabhängig des Grundes kann die IT-Service-Wartung in diesem Fall erst weitergeführt werden, wenn sich die jeweilige IT-Service-Komponente im dafür notwendigen Zustand befindet.

Anti-Pattern: Da Ereignisse sequentiell auftreten, muss das letzte Zustand-Ereignis der zu überprüfenden IT-Service-Komponente vor der Durchführung der Aktivität bestimmt werden. Eine Fehlermeldung wird erzeugt, wenn folgende Bedingung bei der Auswertung eines gerade eintreffenden Vorgang-Ereignisses ve im Ereignis-Strom ES zutrifft:

- Es wird ein Vorgang-Ereignis erkannt, dem eine Zustand-Bedingung zugeordnet ist. Der in der Zustand-Bedingung vorgegebene Zustand für $zapp$ entspricht nicht dem Zustand, der in der Zustand-Ereignis-Historie für app gespeichert ist:

$$\pi_{app,op,prop,value} ve \in \pi_{app,op,prop,value} BM_i \wedge$$
$$\pi_{zapp,status}\left(\sigma_{app=ve.app \wedge op=ve.op \wedge prop=ve.prop \wedge value=ve.value} BM_i\right) /$$
$$\pi_{app,status}(ZEH) \neq \emptyset$$

Als Fehlermeldung wird folgender Text ausgegeben:

Für die Durchführung der Aktivität $\pi_{app,op,prop,value} ve$ müssen zunächst folgende IT-Service-Komponenten in den jeweiligen Status versetzt werden:
$\pi_{zapp,status}(\sigma_{app=ve.app \wedge opp=ve.opp \wedge prop=ve.prop \wedge value=ve.value} BM_i) \,/\, \pi_{app,status}(ZEH)$.

Verwandte Anti-Pattern: Dieses Anti-Pattern ist mit keinem anderen Anti-Pattern verwandt.

5.5 Zusammenfassung des Kapitels

In diesem Kapitel wurden zunächst die Begriffe Pattern und Anti-Pattern definiert. Demnach sind Pattern eine Abstraktion einer bestimmten Form in einem bestimmten Kontext. Anschließend wurde eine Übersicht über bekannte Pattern im Umfeld von betrieblichen Abläufen vorgestellt. Die in dieser Arbeit eingesetzten Pattern sind dabei verwandt zu den in [ETHP14] und [DwAC98] vorgestellten Compliance- und Spezifikationspattern. Um die eingesetzten Pattern detailliert zu beschreiben, wurde zunächst der Kontext, in dem diese angewendet werden, definiert. Anschließend wurden insgesamt zehn Pattern und Anti-Pattern beschrieben. Die vorgestellten Pattern und Anti-Pattern stellen keinen Anspruch auf Vollständigkeit, sondern dienen dazu die im Kapitel zuvor vorgestellten elementaren Schaltmuster, sowie zusätzlich Konfigurationsparameter und Zustand-Bedingungen während einer IT-Service-Wartung überprüfen zu können. Damit sollen die in Kapitel 3.3 vorgestellten Fehler, die während einer IT-Service-Wartung auftreten können, entdeckt werden. Im nachfolgenden Kapitel wird beschrieben, wie diese Pattern auf Basis eines Wartungsplans instanziiert werden.

6 Vorgehen zur IT-gestützten Überwachung von IT-Service-Wartungen mittels Anti-Pattern

Nachdem im vorherigen Kapitel Anti-Pattern zur Entdeckung von Fehlern bei der Durchführung einer IT-Service-Wartung beschrieben wurden, wird in diesem Kapitel das Vorgehen zum Einsatz der in dieser Arbeit entwickelten Methode vorgestellt. Zunächst wird dazu die Erstellung und Nutzung eines Wartungsplans in den allgemeinen Change Management Prozess, wie in Kapitel 2.6 beschrieben, eingebettet. Anschließend wird beschrieben, wie aus einem Wartungsplan die notwendigen Pattern-Instanzen ermittelt werden, um diese für die Überwachung der IT-Service-Wartung zu verwenden.

6.1 Einordnung in das Change Management

Wie in Kapitel 2.6 beschrieben, haben sich in der Praxis folgende Schritte zur Planung und Durchführung einer IT-Service-Wartung im Rahmen des Change Managements etabliert (vgl. [Cisc08, CMAS09, KHWW04, Offi11a]):

1. Formale Beantragung einer geplanten Wartung in Form eines RFC
2. Prüfung des Antrags mit anschließender Freigabe bzw. Ablehnung
3. Erstellung eines Change Plans mit detaillierter Ablaufplanung
4. Prüfung der Ablaufplanung und Freigabe für Tests
5. Durchführung von Tests und Planung der IT-Service-Wartung
6. Durchführung der IT-Service-Wartung und anschließendes Review

Die Erstellung eines Wartungsplans ist methodisch in den dritten Schritt, der Erstellung eines Change Plans, eingebettet. Ein Change Plan beschreibt wie eine geplante IT-Service-Wartung durchzuführen ist. Eine solche Ablaufbeschreibung ist mithilfe der in Kapitel 4.4 vorgestellten Methode zur Modellierung von IT-Service-Wartungen möglich. Nachdem eine erste Version des Wartungsplans erstellt wurde, kann diese im Schritt vier geprüft und für Tests freigegeben werden. Bevor ein Wartungsplan zur Prüfung vorgelegt wird, kann dieser mittels verschiedener analytischer Verfahren auf Korrektheit geprüft werden. Eine Übersicht über verschiedene Analyseverfahren wird in Kapitel 6.2 gegeben. Innerhalb der Tests kann der Wartungsplan überprüft und weiter verfeinert werden. Für die Tests können die

Service Templates der jeweiligen Testumgebung als Markierung im Wartungsplan verwendet werden. Wenn die Durchführung der IT-Service-Wartung in Schritt fünf freigegeben wird, werden im Wartungsplan die Service Templates der Produktionsumgebung verwendet. Vor der Wartungsdurchführung werden die Pattern-Instanzen aus dem Wartungsplan abgeleitet und der Prüfeinheit zur Verfügung gestellt. Die Ableitung von Pattern-Instanzen aus einem Wartungsplan wird in Kapitel 6.4. im Detail beschrieben. Während der eigentlichen Wartungsdurchführung werden die von den IT-Service-Komponenten erzeugten Ereignisse automatisiert an die Prüfeinheit gesendet und gegen die Anti-Pattern überprüft[1]. Wenn die Prüfeinheit eine Anti-Pattern-Instanz erkennt, gibt sie eine entsprechende Fehlermeldung aus. Die Architektur der Prüfeinheit wird in Kapitel 7.2 beschrieben. Abbildung 6.1 stellt den Einsatz des Wartungsplans im Rahmen des Change Managements graphisch dar.

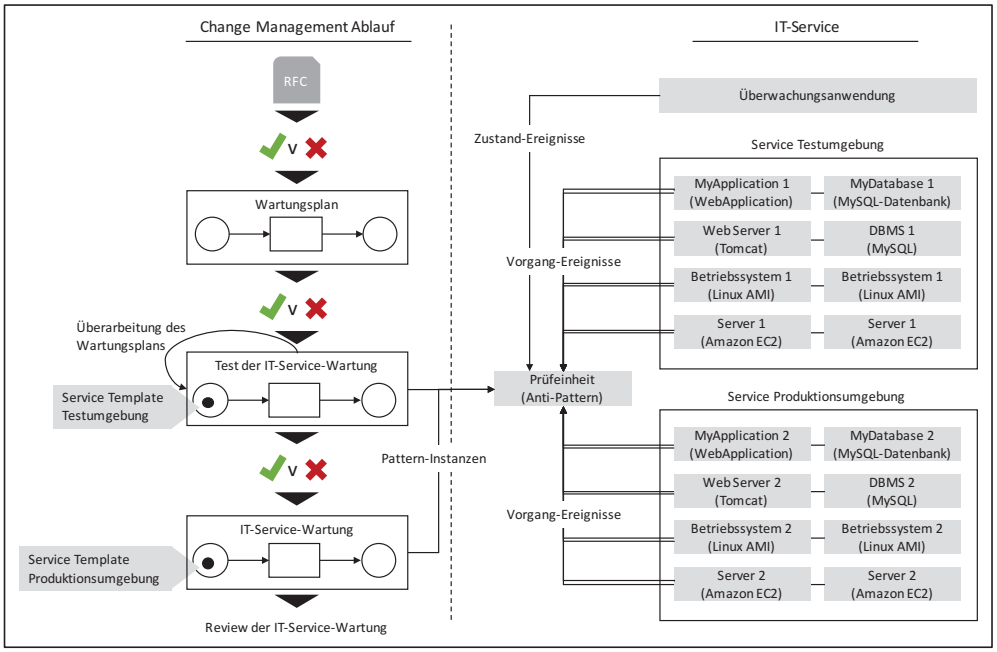


Abbildung 6.1: Einordnung in das Change Management

[1] Es sei angemerkt, dass im Gegensatz zu anderen Arbeiten wie [WZMG11] die Anti-Patterns einmalig in der Prüfeinheit hinterlegt werden und nicht für jeden Wartungsplan neu entwickelt werden. Lediglich die Pattern-Instanzen müssen für jede IT-Service-Wartung ermittelt und der Prüfungseinheit bereitgestellt werden.

6.2 Analyse von Wartungsplänen

Wartungspläne bieten aufgrund ihrer formalen Beschreibung die Möglichkeit, diese zu simulieren und somit ihre erfolgreiche Durchführbarkeit zu validieren. Dadurch kann ein Wartungsplan bereits vor einem ersten Test in einer Testumgebung auf Fehler und seine generelle Durchführbarkeit überprüft werden. Bei der Simulationen mit Petri-Netzen kann zwischen interaktiver und automatischer Simulationen unterschieden werden [Ober96, S.211]. Die interaktive Simulation ist besonders für die Validierung von objekt- und ablaufbezogenen Aspekten geeignet, wohingegen sich die automatisierte Simulation insbesondere für die Überprüfung quantitativer Aspekte eignet [Ober96, S.211]. Für die Simulation von Petri-Netzen wird eine Startmarkierung erzeugt, von der ausgehend Markierungsfolgen erzeugt werden. Der Simulationsverlauf kann dabei in ein Log-File geschrieben werden, um anschließend weitere Analysen durchführen zu können [Ober96, S.214f.]. Eine exemplarische Beschreibung für die Erzeugung von Ablaufmustern wird in Kapitel 6.4.1 gegeben.

Für die Analyse von XML-Netzen lassen sich, aufgrund ihrer Komplexität, bekannte Analysemethoden einfacher Petri-Netz-Typen wie beispielsweise Erreichbarkeitsanalysen, Lebendigkeit und lineare Analysen [Baum97] nicht einfach übertragen [Lenz03]. Um trotzdem Analysemethoden für einfache Petri-Netze nutzen zu können, können Wartungspläne in Stellen/Transitionen-Netze umgewandelt werden, in dem die TOSCA Service Templates entfernt werden. Jedoch kann es passieren, dass die Analyseergebnisse nicht vollständig auf die ursprünglichen Wartungspläne übertragen werden können, da durch das Entfernen der TOSCA Service Templates das Verhalten der Wartungspläne verändert wird [Lenz03]. Ein anderer Ansatz, um Analysemethoden für einfache Petri-Netze auch in höheren Petri-Netzen nutzen zu können, ist das sogenannte „Flattenning“ [EHPB02]. Bei diesem Verfahren wird ein höheres Petri-Netz in ein einfaches Petri-Netz mit demselben Verhalten umgewandelt, wodurch dieses anschließend analysiert werden kann. Allerdings kann eine solche Umwandlung aufgrund des „state explosion problem“ [Valm98] in einem sehr umfangreichen einfachen Petri-Netz mit wesentlich mehr Stellen als beim höheren Petri-Netz resultieren.

6.3 Exkurs: Process Mining

Zur Instanziierung von Pattern auf Basis eines Wartungsplans können Methoden des Process Minings verwendet werden. Process Mining ermöglicht es, auf Basis von ausgeführten Prozessen, ein Modell der Prozessausführung zu erstellen [AaWe04]. Process Mining hat

zum Ziel, die reale Welt und die Modellwelt miteinander zu verknüpfen, in dem eine Beziehung zwischen einfachen realen Ereignissen und Modellelementen erstellt wird [Aals11, S.57]. Dazu werden Ereignis-Logs maschineller Aufgabenträger extrahiert und ausgewertet. Beim Process Mining werden die drei verschiedenen Anwendungsfelder „Entdeckung", „Übereineinstimmung" und „Erweiterung" unterschieden [AAMA11]:

- Bei der „Entdeckung" ist das Ziel, aus Ereignis-Logs ein Ablaufmodell zu rekonstruieren. Vereinfacht ausgedrückt werden dazu in Ereignis-Logs Pattern gesucht und diese Pattern zu einem Ablaufmodell kombiniert.
- Beim Anwendungsfall „Übereinstimmung" wird ein bestehendes Ablaufmodell auf Übereinstimmung mit Ereignis-Logs überprüft. Bei diesem Anwendungstyp können Abweichungen zum Modell entdeckt und analysiert werden.
- Beim Anwendungsfall „Erweiterung" wird ein bestehendes Ablaufmodell auf Basis von Erkenntnissen aus der der Analyse von Ereignis-Logs erweitert.

Für die Instanziierung von Pattern aus einem Wartungsplan ist insbesondere der Anwendungsfall „Entdeckung" von Bedeutung. Der Anwendungsfall „Übereinstimmung" hat von der Idee und den verfolgten Zielen starke Ähnlichkeiten zum in dieser Arbeit präsentierten Ansatz. Beide Anwendungsfälle werden nachfolgend genauer beschrieben.

6.3.1 Entdeckung von Abläufen

Das Ziel bei der Entdeckung von Abläufen ist es, einen Algorithmus zu finden, der ein Ereignis-Log zu einem Ablaufmodell abstrahiert. Hinsichtlich des Ereignis-Logs werden dazu folgende Annahmen getroffen [Aals11, S.99]:

- Ein Ablauf besteht aus Fällen.
- Ein Fall besteht aus Ereignissen, sodass jedes Ereignis einem Fall zugeordnet werden kann.
- Die Ereignisse eines Falls sind geordnet, auch wenn sie parallel ausgeführt wurden.
- Ereignisse haben Attribute, beispielsweise einen Namen, Zeitstempel, den Aufgabenträger, etc.

Die Ereignisse, die zu einem Fall gehören, werden als Ereignis-Pfad repräsentiert. Abbildung 6.2 stellt die Struktur eines beispielhaften Ereignis-Logs dar.

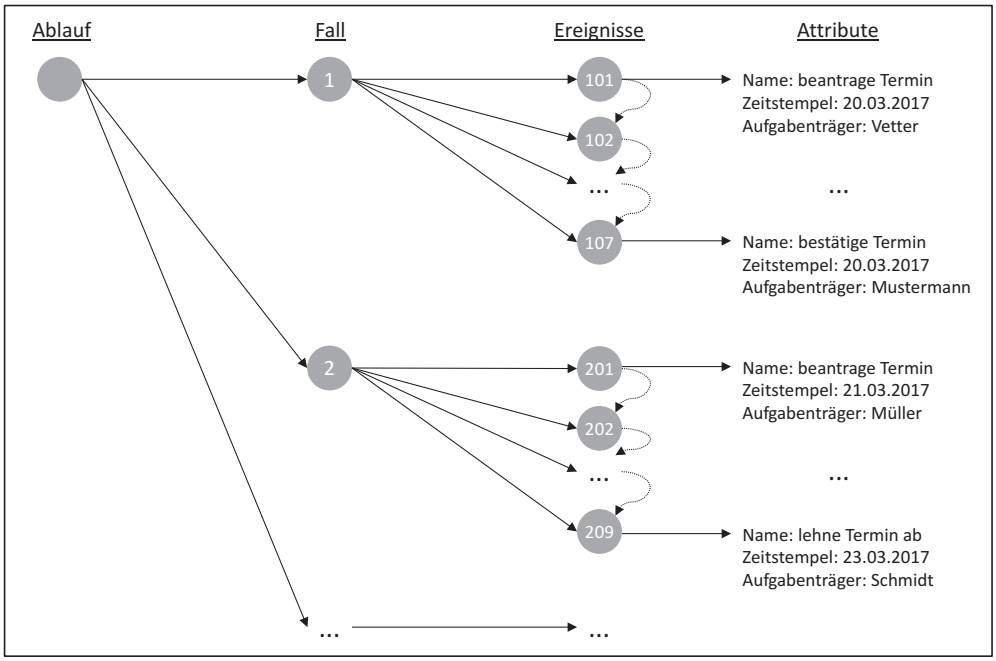


Abbildung 6.2: Struktur eines Ereignis-Logs (angelehnt an [Aals11])

Formal definiert werden ein Ereignis-Pfad und ein Ereignis-Log wie folgt [AaWM04]:

Definition 6.1: Ereignis-Pfad, Ereignis-Log

Sei E eine Menge von Ereignissen. $EP \in E^*$ ist ein Ereignis-Pfad. $L \in O(E^*)$ ist ein Ereignis-Log.[2]

■

Der Algorithmus zur Entdeckung eines Ablaufs ist eine Funktion, die L auf ein Ablaufmodell abstrahiert. Einer der ersten für dieses Ziel entwickelten Algorithmen ist der α-Algorithmus. Der α-Algorithmus verarbeitet ein Ereignis-Log und erstellt daraus ein Workflow-Netz, das „sound“ ist [AaWM04].

Ein Workflow-Netz ist ein Petri-Netz $N = (S, T, F)$, für das gilt [Aals98]:

- Das Petri-Netz besitzt exakt eine einzige Start-Stelle i, für die gilt $\cdot i = \emptyset$ und eine einzige Ende-Stelle o, für die gilt $o \cdot = \emptyset$.

[2] $O(E^*)$ ist die Obermenge von E^*, d.h. $L \subseteq E^*$.

- Wenn zu N eine Transition t^* hinzugefügt wird, die o und i verbindet, d.h. $\cdot t^* = \{o\}$ und $t^* \cdot := \{i\}$, dann ist das resultierende Petri-Netz stark zusammenhängend.

Ein Workflow-Netz ist sound, wenn gilt [Aals98]:

- Für jede aus i erreichbare Markierung M existiert eine Schaltfolge von M nach o, d.h.
$$\forall_M \left(i \overset{*}{\rightarrow} M \right) \Longrightarrow \left(M \overset{*}{\rightarrow} o \right)$$
- Die Markierung o ist die einzige von i erreichbare Markierung mit mindestens einer Marke in o, d.h.
$$\forall_M \left(i \overset{*}{\rightarrow} M \wedge M \geq o \right) \Longrightarrow (M = o)$$
- Es existieren keine toten Transitionen, wenn sich eine Marke in der Start-Stelle befindet, d.h.
$$\forall_{t \in T} \exists_{M,M'}\, i \overset{*}{\rightarrow} M \overset{t}{\rightarrow} M'$$

Viele komplexere Process Mining-Algorithmen bauen auf Methoden des α-Algorithmus auf [Aals11, S.129]. Nachfolgend wird das grundlegende Vorgehen des α-Algorithmus vorgestellt.

Beim α-Algorithmus wird zunächst das Ereignis-Log auf das Auftreten von vier verschiedenen Beziehungsarten zwischen zwei Ereignissen untersucht [AaWM04]. Diese Beziehungsarten sind übernommen aus [AaWM04]:

Definition 6.2: Beziehungsarten

Gegeben sei ein Ereignis-Log L mit $a, b \in L$:

- $a >_L b$, wenn ein Ereignis-Pfad existiert $EP = < e_1, e_2, e_3, \dots, e_n >$ und $i \in \{1, \dots, n-1\}$, so dass $EP \in L$ und $a = e_i$ und $b = e_{i+1}$
- $a \rightarrow_L b$, wenn $a >_L b$ und $b \not> _L a$
- $a \#_L b$, wenn $a \not> _L b$ und $b \not> _L a$
- $a \parallel_L b$, wenn $a >_L b$ und $b >_L a$

■

Angenommen, es liegt ein Ereignis-Log $L_1 = [< acd >, < abcd >, < acbed >]$ vor. Für die Ereignisse in diesem Ereignis-Log ergeben sich folgende Beziehungsarten:

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

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

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

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

Für jedes Ereignis-Log gilt mindestens eine dieser Beziehungsarten für die Ereignis-Paare im Ereignis-Log. Wenn für alle Ereignis-Paare die Beziehungsart ermittelt wurde, lassen sich diese in einer Matrix darstellen. Eine solche Matrix wird als Fußabdruck des Ereignis-Logs bezeichnet [Vand11, S.130]. In Tabelle 6.1 ist der Fußabdruck für das Ereignislog L_1 abgebildet.

Tabelle 6.1: Fußabdruck von L_1

	a	*b*	*c*	*d*	*e*
a	#	→	→	#	#
b	←	#	‖	#	→
c	←	‖	#	→	#
d	#	#	←	#	←
e	#	←	#	→	#

Mithilfe der Beziehungsarten und des daraus erstellten Fußabdrucks lassen sich anschließend die elementaren Ablaufmuster eines Petri-Netzes und somit das gesamte Petri-Netz rekonstruieren. In Abbildung 6.3 sind die elementaren Ablaufmuster und ihre Rekonstruktion aus den Beziehungsarten dargestellt.

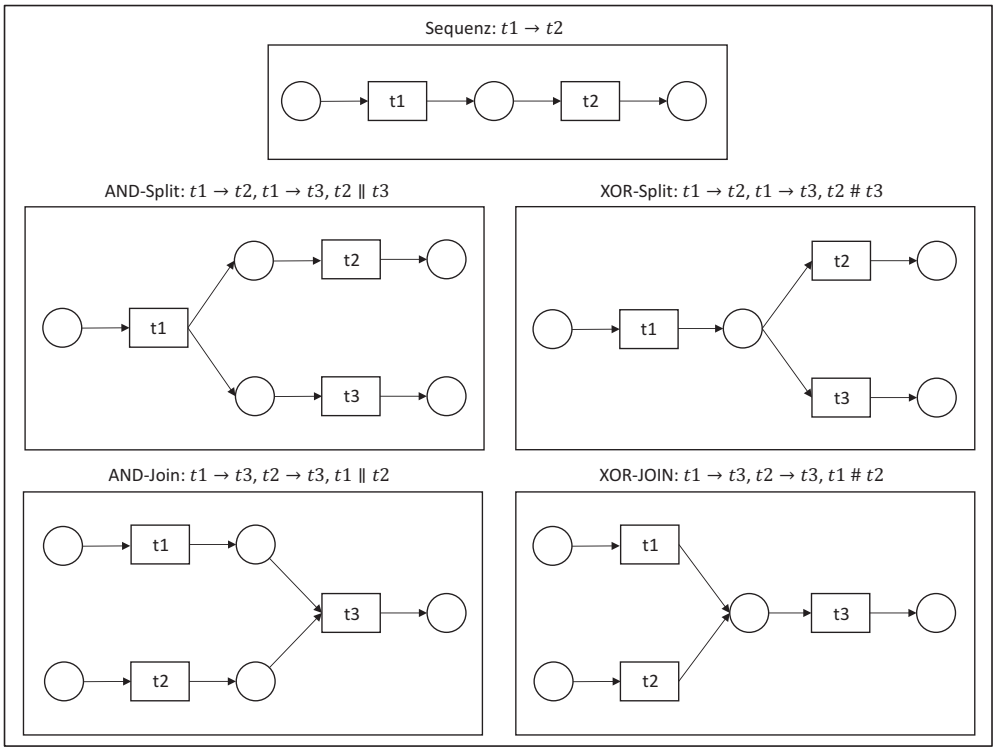


Abbildung 6.3: Elementare Ablaufmuster und ihre Beziehungsarten (angelehnt an [Vand11, S.131])

Bei der Rekonstruktion von Abläufen müssen die vier sich gegenseitig beeinflussenden Eigenschaften

- Übereinstimmung,
- Einfachheit,
- Generalisierung, sowie
- Einschränkung

berücksichtigt werden. Bei der Übereinstimmung ist das Ziel, dass das Ablaufmodel das Ereignis-Log vollständig wiedergeben kann. Dabei ist zu beachten, dass es nicht nur ein Ablaufmodell geben kann, das dazu in der Lage ist. Die Eigenschaft Einfachheit zielt daher darauf ab, aus der Menge möglicher Ablaufmodelle das beste Ablaufmodell zu identifizieren. Das beste Ablaufmodell ist in dieser Hinsicht das einfachste Ablaufmodell, das das Ereignis-Log wiedergeben kann. Für die Entscheidung über das einfachste Ablaufmodell können verschiedene Metriken, wie sie beispielsweise in [MeNA07] beschrieben sind, verwendet werden. Die beiden Eigenschaften Übereinstimmung und Einfachheit alleine

genügen für die Beurteilung eines rekonstruierten Ablaufmodells jedoch nicht, wie am in Abbildung 6.4 dargestellten Ablaufmodell „Blume“ zu sehen. Dieses Modell erfüllt die Eigenschaften Übereinstimmung und Einfachheit für ein Ereignis-Log {a, b, ..., h} vollständig. Jedoch ist es trotzdem kein gutes Modell, da es zu allgemein ist und jeden möglichen Ablauf beschreibt, ohne ihn einzuschränken. Es werden somit mehr Pfade ermöglicht, als sie im Ereignis-Log dokumentiert sind. Das andere Extrem ist ein aufzählendes Modell, in dem alle Pfade im Ereignis-Log als jeweils eigene Sequenz nach einem XOR-Split modelliert werden, wie in Abbildung 6.5 zu sehen. Ein solches Modell ist zu eingeschränkt, so dass bei einem neuen Pfad im Ereignis-Log dieses durch das Modell möglicherweise schon nicht mehr wiedergegeben werden könnte. Ein Algorithmus zur Rekonstruktion von Ablaufmodellen muss daher eine gute Balance zwischen den Eigenschaften Generalisierung und Einschränkung finden.

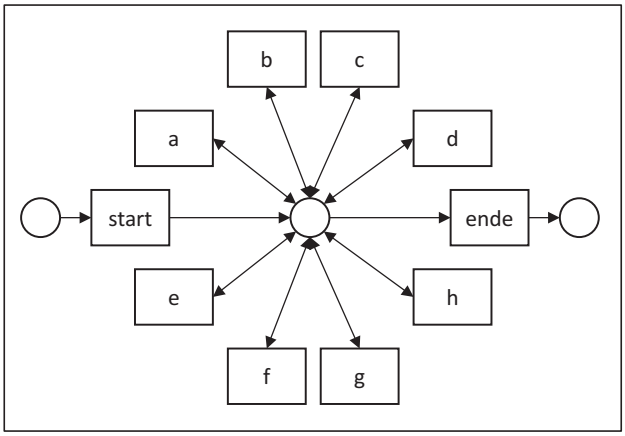


Abbildung 6.4: Ablaufmodell: Blume

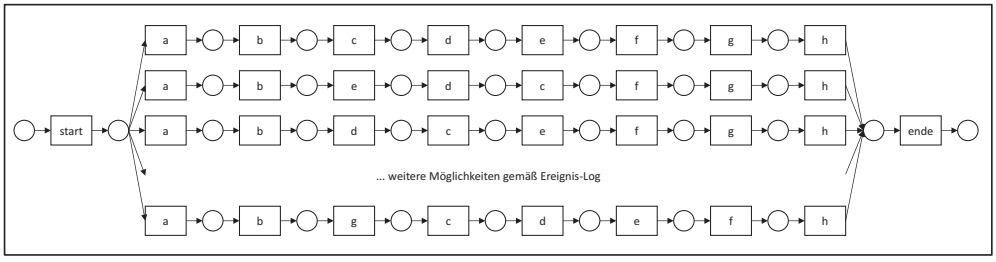


Abbildung 6.5: Ablaufmodell: Aufzählendes Modell

Nachdem die grundlegenden Prinzipien zur Entdeckung von Abläufen aus Ereignis-Logs vorgestellt wurden, wird nachfolgend der Anwendungsfall zur Prüfung von Ereignis-Logs auf Übereinstimmung mit einem bestehenden Ablaufmodell beschrieben.

6.3.2 Übereinstimmung von Abläufen

Zur Überprüfung von Ereignis-Logs auf Übereinstimmung mit Ablaufmodellen können die beiden Methoden „Token Replay“ und der Vergleich von „Fußabdrücken“ verwendet werden. Nachfolgend wird zunächst die Methode Token Replay beschrieben (die Ausführungen in den beiden folgenden Kapiteln sind [Vand11, S.194-208] entnommen).

6.3.2.1 Token Replay

Beim Token Replay werden alle Pfade in einem Ereignis-Log mithilfe eines Petri-Netzes, das sound ist, simuliert, und überprüft in wie vielen Fällen die Ereignisse im Petri-Netz geschaltet werden können. Zum Vergleich der Übereinstimmung wird eine Metrik erstellt, für die die vier Variablen

- p für produzierte Marken,
- k für konsumierte Marken,
- f für fehlende Marken und
- v für verbliebene Marken

verwendet werden. Wenn ein Pfad in einem Ereignis-Log im Ablaufmodel simuliert wird, sind die initialen Belegungen der Variablen $p = k = f = v = 0$ und alle Stellen im Petri-Netz zunächst leer. Anschließend wird das Petri-Netz initial markiert und die Variable p um eins erhöht. Nun werden die Transitionen entsprechend den zugehörigen Ereignissen im Ereignis-Log geschaltet (auch wenn diese gar nicht aktiviert sind). Wenn eine Transition geschaltet wird, wird die Variable p um die Anzahl der durch den Schaltvorgang erzeugten Marken erhöht. Die Variable k wird beim Schaltvorgang um die Anzahl der zu verbrauchenden Marken erhöht. Dabei ist es nicht von Bedeutung, ob im Vorbereich alle benötigten Marken tatsächlich vorhanden sind oder nicht. Es wird stets angenommen, dass alle Marken vorhanden wären. Wenn eine Transition schaltet, jedoch für den Schaltvorgang benötigte Marken im Vorbereich fehlen, wird die Variable f um die Anzahl der im Vorbereich fehlenden Marken erhöht. Wenn Marken nach dem vollständigen Simulieren eines Pfads im Ereignis-Logs im Vorbereich einer Transition verbleiben, wird die Variable v um die Anzahl der verbliebenen Marken erhöht. Die Übereinstimmung eines Pfads σ zum Petri-Netz N wird nun wie folgt berechnet:

$$\text{Übereinstimmung}(\sigma, \text{N}) = \frac{1}{2}\left(1 - \frac{f}{k}\right) + \frac{1}{2}\left(1 - \frac{v}{p}\right)$$

Die Übereinstimmung zwischen einem Pfad σ und einem Petri-Netz N kann demzufolge zwischen 0 und 1 liegen. 1 bedeutet, dass der Pfad und das Petri-Netz vollständig übereinstimmen. 0 bedeutet, dass es überhaupt keine Übereinstimmung zwischen dem Pfad und dem Petri-Netz gibt. Für die Berechnung werden im ersten Teil der Formel die fehlenden Marken in Relation gesetzt zu den konsumierten Marken, d.h. wie viele Marken sollten konsumiert werden und wie viele Marken haben tatsächlich gefehlt. Wenn keine Marken gefehlt haben, gilt:

$$1 - \frac{0}{k} = 1.$$

Im zweiten Teil der Formel werden die übrig gebliebenen Marken in Relation zu den erzeugten Marken gesetzt. Wenn keine Marken übriggeblieben sind, gilt:

$$1 - \frac{0}{p} = 1.$$

Wenn die Übereinstimmung eines ganzen Ereignis-Logs mit einem Ablaufmodell überprüft werden soll, wird dasselbe Verfahren angewendet, nur dass die Summe aller produzierten, konsumierten, verbliebenen und fehlenden Marken der jeweiligen Pfade verwendet wird:

$$\text{Übereinstimmung}(L, \text{N}) = \frac{1}{2}\left(1 - \frac{\Sigma_{\sigma \in L} L(\sigma) \times f_{N,\sigma}}{\Sigma_{\sigma \in L} L(\sigma) \times k_{N,\sigma}}\right) + \frac{1}{2}\left(1 - \frac{\Sigma_{\sigma \in L} L(\sigma) \times v_{N,\sigma}}{\Sigma_{\sigma \in L} L(\sigma) \times p_{N,\sigma}}\right).$$

6.3.2.2 Vergleich des Fußabdrucks

Beim Vergleich von Fußabdrücken wird der Fußabdruck eines Ereignis-Logs mit dem Fußabdruck eines Ablaufmodells verglichen. Um den Fußabdruck eines Ablaufmodells zu generieren, wird zunächst ein Ereignis-Log des Ablaufmodells erzeugt. Dieses kann beispielsweise durch die Simulation des Ablaufmodells erzeugt werden. Anschließend wird nach dem Verfahren wie in Kapitel 6.3.1 beschrieben der Fußabdruck des erzeugten Ereignis-Logs erstellt. Die einzelnen Zellen der beiden Fußabdrücke werden nun miteinander verglichen. Beispielhaft soll der Fußabdruck in Tabelle 6.1 mit dem Fußabdruck in Tabelle 6.2 verglichen werden, der den Fußabdruck des zu vergleichenden Ablaufmodells darstellt.

Tabelle 6.2: Fußabdruck eines Ablaufmodells

	a	*b*	*c*	*d*	*e*
a	#	→	→	→	#
b	←	#	→	#	→
c	←	←	#	→	#
d	←	#	←	#	←
e	#	←	#	→	#

Tabelle 6.3 stellt die Differenzen in den beiden Fußabdrücken dar. Demnach unterscheiden sich die Zellen (a, d), (d, a), (b, c), (c, b) in den beiden Fußabdrücken, da im Fußabdruck des Ablaufmodells beispielsweise das Ereignis c nach dem Ereignis b ausgeführt wird. Im Fußabdruck des zu vergleichenden Ereignis-Logs werden die Ereignisse b und c jedoch nebenläufig ausgeführt.

Tabelle 6.3: Differenz-Fußabdruck

	a	*b*	*c*	*d*	*e*
a				#:→	
b			‖: →		
c		‖: ←			
d	#:←				
e					

6.4 Instanziierung von Pattern

Nachdem im vorangegangen Kapitel die wesentlichen Grundlagen zum Process Mining, wie sie nachfolgend benötigt werden, beschrieben wurden, wird in diesem Kapitel die Instanziierung der in Kapitel 5.4 vorgestellten Pattern aus einem Wartungsplan beschrieben. Die Instanziierung lässt sich dabei in folgende vier Schritte zusammenfassen:

1. Simulieren des Wartungsplans zur Erzeugung eines Ereignis-Logs für ein Service Template
2. Erstellen eines Fußabdrucks zur Erzeugung der Instanzen für die Pattern eins bis vier
3. Erstellen eines Fußabdrucks zur Erzeugung der Instanzen für die Pattern fünf und sechs
4. Erstellung der Instanzen für die Pattern sieben bis zehn

Wenn im Rahmen eines Wartungsplans mehrere IT-Services gewartet werden sollen, d.h. der Wartungsplan in der Startmarkierung mehr als ein Service Template enthält, werden

die Schritte eins bis vier wiederholt, bis alle Service Templates simuliert sind. Nachfolgend werden die einzelnen Schritte im Detail beschrieben.

6.4.1 Erzeugung eines Ereignis-Logs

Um ein Ereignis-Log für einen Wartungsplan zu erzeugen, muss der Wartungsplan simuliert werden. Dazu werden, wie von Oberweis [Ober96, S.211 ff.] beschrieben, Markierungsfolgen erzeugt. Die Erzeugung von Markierungsfolgen wird am Beispiel des Wartungsplans in Abbildung 6.6 und dem beispielhaften Service Template in Quelltext 5 beschrieben. Im Service Template sind, zur Veranschaulichung wie mehrere Aktivitäten beim Schalten einer Transition ausgeführt werden, zwei Node Templates desselben Node Types enthalten.

```
1  <ServiceTemplate id="MyERP">
2  …
3  <NodeTemplate id="MyWebServer" name="WebServer" nodeType="WebServer">
4   <Properties>
5    <listen>admin</listen>
6  ...
7   </Properties>
8   ...
9  </NodeTemplate>
10 <NodeTemplate id="MyDatabase_1" name="MyDatabase_1" nodeType="Data-
   base">
11  <Properties>
12   <port>841</port>
13 ...
14  </Properties>
15  ...
16 </NodeTemplate>
17 <NodeTemplate id="MyDatabase_2" name="MyDatabase_2" nodeType="Data-
   base">
18  <Properties>
19   <port>841</port>
20 ...
21  </Properties>
22  ...
23 </NodeTemplate>
24 <NodeTemplate id="MyAppServer" name="MyAppServer" nodeType="AppS-
   erver">
25  <Properties>
26   <connectiontimeout>10</ connectiontimeout>
27 ...
28  </Properties>
29  ...
30 </NodeTemplate>
31 ...
32 </ServiceTemplate>
```

Quelltext 6.1: Ausschnitt eines Service Templates

Ausgehend von einer Startmarkierung $(\{1\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\})$[3] ist die Transition t1 aktiviert und wird geschaltet. Ein Schaltvorgang wird durch die Transition und die Aktivitäten, die durch das Service Template in der Vorstelle der Transition initiiert sind, identifiziert. Durch den Schaltvorgang ändert sich die Markierung zu $(\{\},\{1\},\{1\},\{\},\{\},\{\},\{\},\{\},\{\},\{\})$ und die beiden Transitionen t2 und t3 sind aktiviert. Nun wird eine der beiden aktivierten Transitionen geschaltet, beispielsweise t2. Dadurch ändert sich die Markierung zu $(\{\},\{\},\{1\},\{1\},\{\},\{\},\{\},\{\},\{\},\{\})$. Im Gegensatz zu [Ober96, S.212f.] können Transitionen nicht gleichzeitig schalten, sondern es kann zu einem bestimmten Zeitpunkt immer nur eine einzige Transition schalten, auch wenn mehrere Transitionen gleichzeitig aktiviert sind. Eine solche Ausführung ist nicht effizient, allerdings wird dadurch sichergestellt, dass im Ereignislog stets eine eindeutige Aussage getroffen werden kann, welche Transition vor oder nach einer anderen Transition stattgefunden hat. Die Entscheidung welche Transition schaltet, wenn gleichzeitig mehrere Transitionen aktiviert sind, trifft bei der interaktiven Simulation der Nutzer, der die Simulation ausführt. Bei der automatisierten Simulation ist sicherzustellen, dass bei mehreren gleichzeitig aktivierten Transitionen jede Transition mindestens einmal vor den anderen gleichzeitig aktivierten Transitionen schaltet. In welcher Reihenfolge die Transitionen schalten kann zufällig entschieden werden. Ein Pfad ist komplett simuliert, wenn keine weiteren Markierungsfolgen mehr möglich sind. In Abbildung 6.6 wäre dies unter der Annahme, dass jede Transition schalten kann, bei der Markierungsfolge $(\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{1\})$ der Fall.

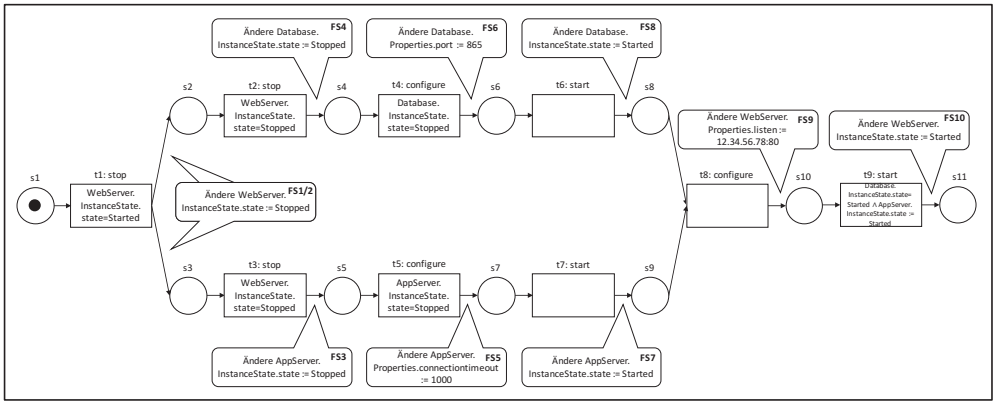


Abbildung 6.6: Beispielhafter Wartungsplan

[3] Analog zu [Ober96, S.212] wird eine Markierung als Vektor dargestellt. Die Komponenten des Vektors sind Tupelmengen und werden, am Beispiel der Startmarkierung, wie folgt geordnet: $(M^0(p1), M^0(p2), M^0(p3), M^0(p4), M^0(p5), M^0(p6), M^0(p7), M^0(p8), M^0(p8), M^0(p10))$.

Durch das wiederholte Schalten von Transitionen werden Markierungsfolgen erzeugt. Tabelle 6.4 zeigt zwei mögliche Markierungsfolgen für den Wartungsplan aus Abbildung 6.6. Der Pfeil $\downarrow tx[a_i, \ldots, a_n]$ zwischen zwei Markierungen zeigt an, welche Transition geschaltet werden muss, um die nächste Markierung zu erreichen, und welche Aktivitäten dadurch ausgeführt werden. Beim Schalten von $t2$ werden beispielsweise zwei Aktivitäten ausgeführt, da das Filterschema auf zwei Node Templates im Service Template (MyDatabase_1 und MyDatabase_2) angewendet wird. Für die beiden Markierungsfolgen in Tabelle 6.4 ergeben sich somit die Pfade

$< t1[a_1], t2[a_2, a_3], t4[a_4, a_5], t6[a_6, a_7], t3[a_8], t5[a_9], t7[a_{10}], t8[a_{11}], t9[a_{12}] >$ und
$< t1[a_1], t2[a_2, a_3], t3[a_8], t5[a_9], t4[a_4, a_5], t7[a_{10}], t6[a_6, a_7], t8[a_{11}], t9[a_{12}] >$.

Tabelle 6.4: Markierungsfolgen

Markierungsfolge 1	Markierungsfolge 2
$(\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\})$	$(\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\})$
$\downarrow t1[a_1]$	$\downarrow t1[a_1]$
$(\{\},\{1\},\{1\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\})$	$(\{\},\{1\},\{1\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\})$
$\downarrow t2[a_2, a_3]$	$\downarrow t2[a_2, a_3]$
$(\{\},\{\},\{1\},\{1\},\{\},\{\},\{\},\{\},\{\},\{\},\{\})$	$(\{\},\{\},\{1\},\{1\},\{\},\{\},\{\},\{\},\{\},\{\},\{\})$
$\downarrow t4[a_4, a_5]$	$\downarrow t3[a_8]$
$(\{\},\{\},\{1\},\{\},\{\},\{1\},\{\},\{\},\{\},\{\},\{\})$	$(\{\},\{\},\{\},\{1\},\{1\},\{\},\{\},\{\},\{\},\{\},\{\})$
$\downarrow t6[a_6, a_7]$	$\downarrow t5[a_9]$
$(\{\},\{\},\{1\},\{\},\{\},\{\},\{\},\{1\},\{\},\{\},\{\})$	$(\{\},\{\},\{\},\{1\},\{\},\{\},\{1\},\{\},\{\},\{\},\{\})$
$\downarrow t3[a_8]$	$\downarrow t4[a_4, a_5]$
$(\{\},\{\},\{\},\{\},\{1\},\{\},\{\},\{1\},\{\},\{\},\{\})$	$(\{\},\{\},\{\},\{\},\{\},\{\},\{1\},\{1\},\{\},\{\},\{\},\{\})$
$\downarrow t5[a_9]$	$\downarrow t7[a_{10}]$
$(\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{1\},\{1\},\{\},\{\},\{\})$	$(\{\},\{\},\{\},\{\},\{\},\{\},\{1\},\{\},\{\},\{1\},\{\},\{\})$
$\downarrow t7[a_{10}]$	$\downarrow t6[a_6, a_7]$
$(\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{1\},\{1\},\{\},\{\})$	$(\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{1\},\{1\},\{\},\{\})$
$\downarrow t8[a_{11}]$	$\downarrow t8[a_{11}]$
$(\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{1\},\{\})$	$(\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{1\},\{\})$
$\downarrow t9[a_{12}]$	$\downarrow t9[a_{12}]$
$(\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{1\})$	$(\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{\},\{1\})$

Wenn alle Markierungsfolgen für einen Wartungsplan erzeugt sind, kann daraus das Ereignis-Log für den jeweiligen Wartungsplan erstellt werden. Dabei entspricht jede Markierungsfolge mindestens einem Pfad im Ereignis-Log und jede ausgeführte Aktivität einem Ereignis. Da mehrere Aktivitäten gleichzeitig in einem Schaltvorgang ausgeführt werden können, ergeben sich daraus mehrere Pfade im Ereignis-Log. Aktivitäten die gleichzeitig geschaltet werden, müssen mindestens einmal in umgekehrter Reihenfolge im Ereignis-Log auftreten. Tabelle 6.5 zeigt ein beispielhaftes Ereignis-Log für die Markierungsfolge 1 aus Tabelle 6.4.

Tabelle 6.5: Beispielhaftes Ereignis-Log

Fall	st	app	op	prop	value	start	ende
1	MyERP	MyWebServer	stop	Null	Null	15.07.17 11:00:00	15.07.17 11:30:00
	MyERP	MyDatabase_1	stop	Null	Null	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyDatabase_2	stop	Null	Null	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyDatabase_1	configure	port	865	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyDatabase_2	configure	port	865	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyDatabase_1	start	Null	Null	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyDatabase_2	start	Null	Null	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyAppServer	stop	Null	Null	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyAppServer	configure	connection timeout	1000	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyAppServer	start	Null	Null	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyWebServer	configure	listen	12.34.56.78:80	15.07.17 12:00:01	15.07.17 12:30:00
	MyERP	MyWebServer	start	Null	Null	15.07.17 12:00:01	15.07.17 12:30:00

2	MyERP	MyWebServer	stop	Null	Null	15.07.17 11:00:00	15.07.17 11:30:00
	MyERP	MyDatabase_2	stop	Null	Null	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyDatabase_1	stop	Null	Null	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyDatabase_2	configure	port	865	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyDatabase_1	configure	port	865	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyDatabase_2	start	Null	Null	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyDatabase_1	start	Null	Null	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyAppServer	stop	Null	Null	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyAppServer	configure	connection timeout	1000	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyAppServer	start	Null	Null	15.07.17 11:30:01	15.07.17 12:00:00
	MyERP	MyWebServer	configure	listen	12.34.56.78:80	15.07.17 12:00:01	15.07.17 12:30:00
	MyERP	MyWebServer	start	Null	Null	15.07.17 12:00:01	15.07.17 12:30:00

6.4.2 Instanziierung der Pattern eins bis vier

Zur Instanziierung der Pattern NACHFOLGER, DIREKTER NACHFOLGER, VORGÄNGER und DIREKTER VORGÄNGER werden die Ereignisse im Ereignis-Log des zu überwachenden Wartungsplans zunächst auf die in Definition 6.2 beschriebenen Beziehungsarten untersucht. Daraus wird ein Fußabdruck erzeugt. Für jedes Service Template wird ein separater Fußabdruck erstellt. Tabelle 6.6 zeigt den Fußabdruck des Wartungsplans in Abbildung 6.6.

Tabelle 6.6: Fußabdruck des Wartungsplans in Abbildung 6.6

	$t1[a_1]$	$t2[a_2]$	$t2[a_3]$	$t3[a_8]$	$t4[a_4]$	$t4[a_5]$	$t5[a_9]$	$t6[a_6]$	$t6[a_7]$	$t7[a_{10}]$	$t8[a_{11}]$	$t9[a_{12}]$
$t1[a_1]$	#	→	→	→	#	#	#	#	#	#	#	#
$t2[a_2]$	←	#	‖	‖	→	→	‖	#	#	‖	#	#
$t2[a_3]$	←	‖	#	‖	→	→	‖	#	#	‖	#	#
$t3[a_8]$	←	‖	‖	#	‖	‖	→	‖	‖	#	#	#
$t4[a_4]$	#	←	←	‖	#	‖	‖	→	→	‖	#	#
$t4[a_5]$	#	←	←	‖	‖	#	‖	→	→	‖	#	#
$t5[a_9]$	#	‖	‖	←	‖	‖	#	‖	‖	→	#	#
$t6[a_6]$	#	#	#	‖	←	←	‖	#	‖	‖	→	#
$t6[a_7]$	#	#	#	‖	←	←	‖	‖	#	‖	→	#
$t7[a_{10}]$	#	‖	‖	#	‖	‖	←	‖	‖	#	→	#
$t8[a_{11}]$	#	#	#	#	#	#	#	←	←	←	#	→
$t9[a_{12}]$	#	#	#	#	#	#	#	#	#	#	←	#

Anhand des Fußabdrucks für ein Ereignis-Log L mit den Ereignissen $\{e_1, e_2, e_3, \dots, e_n\} \in L$ und $i \in \{1, \dots, n\}, j \in \{1, \dots, n\}, k \in \{1, \dots, n\}$ lassen sich die Pattern nun, wie in Tabelle 6.7 beschrieben, ableiten.

Tabelle 6.7: Übersicht Instanziierung Pattern 1-4

NACHFOLGER[4]	**DIREKTER NACHFOLGER**
• wenn $e_i \rightarrow e_j$ und $e_i \rightarrow e_k$ und $e_j \parallel e_k$ oder • wenn $e_i \rightarrow e_j$und es gibt mindestens einmal $e_i \parallel e_k$ und $e_j \parallel e_k$	• wenn $e_i \rightarrow e_j$ und $e_i \rightarrow e_k$ und $e_j \# e_k$ oder • wenn $e_i \rightarrow e_j$ und es gibt kein $e_i \parallel e_k$ und kein $e_j \parallel e_k$
VORGÄNGER	**DIREKTER VORGÄNGER**
• wenn $e_i \leftarrow e_j$ und $e_k \leftarrow e_j$ und $e_i \parallel e_k$ oder • wenn $e_i \leftarrow e_j$und es gibt mindestens einmal $e_i \parallel e_k$ und $e_j \parallel e_k$	• wenn $e_i \leftarrow e_j$ und $e_k \leftarrow e_j$ und $e_i \# e_k$ oder • wenn $e_i \leftarrow e_j$und es gibt kein $e_i \parallel e_k$ und kein $e_j \parallel e_k$

[4] Die Pattern-Instanzen für NACHFOLGER werden in diesem Schritt zunächst nur für (a_{akt}, a_{nach}) ermittelt.

Für den Fußabdruck in Tabelle 6.6 ergeben sich somit folgende Pattern-Instanzen für das Ereignis-Log L:

NACHFOLGER = $\{(a_1, a_2, NULL), (a_1, a_3, NULL), (a_1, a_8, NULL), (a_2, a_4, NULL), (a_2, a_5, NULL), (a_3, a_4, NULL), (a_3, a_5, NULL), (a_4, a_6, NULL), (a_4, a_7, NULL), (a_5, a_6, NULL), (a_5, a_7, NULL), (a_6, a_{11}, NULL), (a_7, a_{11}, NULL), (a_8, a_9, NULL), (a_9, a_{10}, NULL), (a_{10}, a_{11}, NULL)\}$

DIREKTER NACHFOLGER = $\{(a_{11}, a_{12})\}$

VORGÄNGER = $\{(a_2, a_1), (a_3, a_1), (a_8, a_1), (a_4, a_2), (a_4, a_3), (a_5, a_2), (a_5, a_3), (a_6, a_4), (a_6, a_5), (a_7, a_4), (a_7, a_5), (a_9, a_8), (a_{10}, a_9), (a_{11}, a_6), (a_{11}, a_7), (a_{11}, a_{10}), (a_{12}, a_{11})\}$

DIREKTER VORGÄNGER = $\{(a_{12}, a_{11})\}$

Für das Pattern NACHFOLGER reicht ein solcher Fußabdruck nicht aus, da damit im Tupel $(a_{akt}, a_{nach}, a_{entf})$ a_{entf} nicht ermittelt werden kann. Hierfür werden erweiterte Beziehungsarten benötigt, mit denen alle nachfolgenden Aktivitäten zu einer bestimmten Aktivität in einem Ereignislog identifiziert werden können, und nicht nur Aktivitäten, die im Ereignislog immer direkt nach einer Aktivität ausgeführt werden. Außerdem können mit diesen erweiterten Beziehungsarten auch alle Aktivitäten identifiziert werden, die niemals gemeinsam mit einer anderen Aktivität in einem Pfad auftauchen. Dies wird zur Instanziierung der Pattern ALTERNATIVE ANWESENHEIT und ALTERNATIVE ABWESENHEIT benötigt und in Kapitel 6.4.3 bzw. 6.4.4 beschrieben. Die erweiterten Beziehungsarten sind [WeMW09] entnommen und wie folgt definiert.

Definition 6.3 Erweiterte Beziehungsarten

Gegeben sei ein Ereignis-Log L mit $a, b \in L$:

- $a \succ_L b$, wenn ein Ereignis-Pfad existiert $EP = < e_1, e_2, e_3, \dots, e_n >$, $i \in \{1, \dots, n-1\}$ und $i < j \leq n$, so dass $EP \in L$ und $a = e_i$ und $b = e_j$
- $a \perp_L b$, wenn $a \not\succ_L b$ und $b \not\succ_L a$
- $a \rightsquigarrow_L b$, wenn $a \succ_L b$ und $b \not\succ_L a$

■

Tabelle 6.8 zeigt den Fußabdruck mit den erweiterten Beziehungsarten für den Wartungsplan in Abbildung 6.6.

Tabelle 6.8: Fußabdruck des Wartungsplans in Abbildung 6.6 mit erweiterten Beziehungsarten

	$t1[a_1]$	$t2[a_2]$	$t2[a_3]$	$t3[a_8]$	$t4[a_4]$	$t4[a_5]$	$t5[a_9]$	$t6[a_6]$	$t6[a_7]$	$t7[a_{10}]$	$t8[a_{11}]$	$t9[a_{12}]$
$t1[a_1]$		⇝	⇝	⇝	⇝	⇝	⇝	⇝	⇝	⇝	⇝	⇝
$t2[a_2]$					⇝	⇝		⇝	⇝		⇝	⇝
$t2[a_3]$					⇝	⇝		⇝	⇝		⇝	⇝
$t3[a_8]$							⇝			⇝	⇝	⇝
$t4[a_4]$								⇝	⇝		⇝	⇝
$t4[a_5]$								⇝	⇝		⇝	⇝
$t5[a_9]$										⇝	⇝	⇝
$t6[a_6]$											⇝	⇝
$t6[a_7]$											⇝	⇝
$t7[a_{10}]$											⇝	⇝
$t8[a_{11}]$												⇝
$t9[a_{12}]$												

a_{entf} wird nun entsprechend für alle e_k in den Tupeln (e_i, e_j, e_k) instanziiert, für die gilt, dass $(e_j \rightsquigarrow e_k)$.

Somit ergeben sich für das Pattern NACHOLFGER folgende Pattern-Instanzen:

NACHFOLGER = $\{(a_1, a_2, a_4), (a_1, a_2, a_5), (a_1, a_2, a_6), (a_1, a_2, a_7), (a_1, a_2, a_{11}),$
$(a_1, a_2, a_{12}), (a_1, a_3, a_4), (a_1, a_3, a_5), (a_1, a_3, a_6), (a_1, a_3, a_7), (a_1, a_3, a_{11}), (a_1, a_3, a_{12}),$
$(a_1, a_8, a_9), (a_1, a_8, a_{10}), (a_1, a_8, a_{11}), (a_1, a_8, a_{12}), (a_2, a_4, a_6), (a_2, a_4, a_7), (a_2, a_4, a_{11}),$
$(a_2, a_4, a_{12}), (a_2, a_5, a_6), (a_2, a_5, a_7), (a_2, a_5, a_{11}), (a_2, a_5, a_{12}), (a_3, a_4, a_6), (a_3, a_4, a_7),$
$(a_3, a_4, a_{11}), (a_3, a_4, a_{12}), (a_3, a_5, a_6), (a_3, a_5, a_7), (a_3, a_5, a_{11}), (a_3, a_5, a_{12}), (a_4, a_6, a_{11}),$
$(a_4, a_6, a_{12}), (a_4, a_7, a_{11}), (a_4, a_7, a_{12}), (a_5, a_6, a_{11}), (a_5, a_6, a_{12}), (a_5, a_7, a_{11}),$
$(a_5, a_7, a_{12}), (a_6, a_{11}, a_{12}), (a_7, a_{11}, a_{12}), (a_8, a_9, a_{10}), (a_8, a_9, a_{11}), (a_8, a_9, a_{12}),$
$(a_9, a_{10}, a_{11}), (a_9, a_{10}, a_{12}), (a_{10}, a_{11}, a_{12})\}$

6.4.3 Instanziierung der Pattern fünf und sechs

Zur Instanziierung der Pattern ANWESENHEIT und ALTERNATIVE ANWESENHEIT wird ebenfalls je Service Template ein Fußabdruck mit den erweiterten Beziehungsarten aus Definition 6.3 für das Ereignis-Log des zu überprüfenden Wartungsplans erzeugt.

Tabelle 6.9 zeigt den Fußabdruck für das beispielhafte Ablaufmodell in Abbildung 6.7 unter Einsatz der erweiterten Beziehungsarten. Für das Beispiel wird angenommen, das beim Schalten einer Transaktion exakt eine Aktivität ausgeführt wird.

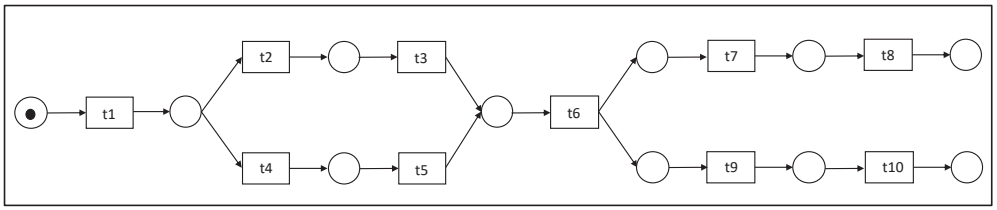


Abbildung 6.7: Netz zur Illustration erweiterter Beziehungsarten

Tabelle 6.9: Fußabdruck des Ablaufmodells in Abbildung 6.7 mit erweiterten Beziehungsarten

	$t1[a_1]$	$t2[a_2]$	$t3[a_3]$	$t4[a_4]$	$t5[a_5]$	$t6[a_6]$	$t7[a_7]$	$t8[a_8]$	$t9[a_9]$	$t10[a_{10}]$
$t1[a_1]$		⇝	⇝	⇝	⇝	⇝	⇝	⇝	⇝	⇝
$t2[a_2]$			⇝	⊥	⊥	⇝	⇝	⇝	⇝	⇝
$t3[a_3]$				⊥	⊥	⇝	⇝	⇝	⇝	⇝
$t4[a_4]$		⊥	⊥		⇝	⇝	⇝	⇝	⇝	⇝
$t5[a_5]$		⊥	⊥			⇝	⇝	⇝		
$t6[a_6]$							⇝	⇝	⇝	⇝
$t7[a_7]$								⇝		
$t8[a_8]$										
$t9[a_9]$										⇝
$t10[a_{10}]$										

Anhand des Fußabdrucks für ein Ereignis-Log L mit den Ereignissen $\{e_1, e_2, e_3, \dots, e_n\} \in L$ und $i \in \{1, \dots, n\}$, $j \in \{1, \dots, n\}$ wird das Pattern ALTERNATIVE ANWESENHEIT wird mit allen Ereignispaaren (e_i, e_j) instanziiert für die gilt, dass $e_i \perp e_j$.

Das Pattern ANWESENHEIT wird mit allen Ereignissen e_i instanziiert für die gilt, dass sie in jedem Ereignis-Pfad EP in L auftreten.

Für den Fußabdruck in Tabelle 6.9 ergeben sich somit folgende Pattern-Instanzen für das Ereignis-Log L:

ANWESENHEIT = $\{a_1, a_7, a_8, a_9, a_{10}\}$

ALTERNATIVE ANWESENHEIT = $\{(a_2, a_4), (a_2, a_5), (a_3, a_4), (a_3, a_5), (a_4, a_2), (a_4, a_3), (a_5, a_2), (a_5, a_3)\}$

6.4.4 Instanziierung der Pattern sieben bis zehn

Die Instanziierung des Patterns ABWESENHEIT ist für das entsprechende Anti-Pattern nicht notwendig, da für die Erkennung von Anti-Pattern-Instanzen die Menge aller geplanten Aktivitäten AM ausreichend ist (siehe Kapitel 5.4.7). AM wird initiiert durch jede Aktivität, die in einem Ereignispfad in einem Ereignislog L auftritt. Die Instanziierung des Patterns ALTERNATIVE ABWESENHEIT entspricht der Instanziierung des Patterns ALTERNATIVE ANWESENHEIT und wird daher an dieser Stelle nicht weiter beschrieben. Für die Instanziierung des Patterns WERT wird ebenfalls die Menge AM benötigt, die anschließend entsprechend selektiert wird, um nur die Aktivitäten zu erhalten, mit denen ein Property eines Node Templates geändert wird[5]. Für die Instanziierung des Patterns ZUSTAND werden über die Funktionen $B: BM \rightarrow TIM$ und $TI: T \rightarrow \Psi$ alle Zustand-Bedingungen für den Wartungsplan ermittelt.

6.5 Zusammenfassung des Kapitels

In diesem Kapitel wurde zunächst die Einordnung der in dieser Arbeit vorgestellten Methode zum Einsatz im Change Management Prozess, wie er in Kapitel 2.6 beschrieben wurde, vorgenommen. Anschließend wurden Möglichkeiten beschrieben, um Wartungspläne a priori auf ihre Durchführbarkeit zu analysieren. Da Wartungspläne den Soll-Prozess der Wartungsdurchführung beschreiben und als Grundlage zur Instanziierung von Pattern dienen, ist es wichtig, dass der Wartungsplan selbst korrekt ist. Für Wartungspläne eignet sich zur Analyse insbesondere die Simulation, da Analysemethoden einfacher Petri-Netze nicht einfach auf höhere Petri-Netze angewendet werden können. Die Simulation ist zudem Voraussetzung zur Instanziierung der in Kapitel 5.4 vorgestellten Pattern. Das durch die Simulation erzeugte Log eines Wartungsplans wird verwendet, um daraus einen Fußabdruck zu erzeugen. Mithilfe des Fußabdrucks können alle Pattern zur Überprüfung des Kontrollflusses instanziiert werden. Die Erzeugung eines Fußbadrucks aus einem Simulationslog ist zentraler Bestandteil vieler Process Mining Methoden. Deshalb wurden in einem Exkurs die Grundlagen des Process Minings vorgestellt und die in dieser Arbeit entwickelte Methode zu den Anwendungsfällen des Process Minings eingeordnet.

[5] Eine Aktivität, bei der kein Property eines Node Templates geändert wird, kann zum Beispiel die Änderung des Status eines Node Templates sein (z.B. wenn eine IT-Service-Komponente gestartet oder gestoppt werden soll).

7 Architektur und Konzeption eines Prototyps

Nachdem im vorhergehenden Kapitel die Vorgehensweise zur Anwendung der in dieser Arbeit entwickelten Methode im Rahmen des Change Managements beschrieben wurde, wird in diesem Kapitel eine Architektur und eine prototypische Implementierung zur IT-gestützten Anwendung der Methode vorgestellt. Dazu werden zunächst die Anforderungen an eine solche Architektur und anschließend die Architektur selbst beschrieben. Ein besonderes Augenmerk wird dabei auf die Prüfeinheit gelegt, die die eintreffenden Ereignisse gegen die Anti-Pattern prüft. Im letzten Abschnitt wird eine prototypische Implementierung der Architektur vorgestellt.

7.1 Anforderungen

Um den Prototyp zur Überwachung von IT-Service-Wartungen einsetzen zu können, werden basierend auf der Motivation in Kapitel 1 folgende Anforderungen an die zu überwachenden IT-Service-Komponenten gestellt:

1. Jede Wartungsaktivität an einer IT-Service-Komponente, wie beispielsweise das Starten, Stoppen oder Konfigurieren der IT-Service-Komponente, muss in ein Log geschrieben werden.
2. Jeder Eintrag in einem Log muss mindestens die Elemente $timestamp$, op, $prop$ und $value$ beinhalten.
3. Alle zu überwachenden IT-Service-Komponenten nutzen synchronisierte Uhren, d.h. die generierten Zeitstempel in den Log-Einträgen werden auf Basis einer gemeinsamen übergeordneten Zeitmessung erstellt.
4. Die Einträge im Log müssen in einem von der Prüfeinheit lesbaren Format (z.B. JSON, XML) direkt an diese gesendet werden können. Der Versand der Einträge an die Prüfeinheit soll umgehend nach deren Erstellung erfolgen, so dass diese zeitlich geordnet in der Prüfeinheit eintreffen.

An die Architektur werden zudem folgende Anforderungen gestellt:

1. Die Überprüfung der eintreffenden Ereignisse soll online erfolgen, damit bereits während der Durchführung der IT-Service-Wartung mögliche Wartungsfehler erkannt werden.
2. Für die Modellierung von Wartungsplänen soll eine graphische Benutzungsoberfläche zur Verfügung stehen.
3. Die Pattern-Instanzen sollen automatisiert erstellt werden können und in einem von der Prüfeinheit lesbaren Format (z.B. JSON, XML) an diese übergeben werden können.
4. Erzeugte Fehlermeldungen müssen dem IT-Service-Anbieter, der die Durchführung der IT-Service-Wartung überwacht, für ihn verständlich angezeigt werden.

Für die Umsetzung dieser Anforderungen wird im nachfolgenden Kapitel die Architektur beschrieben, die bei der Erstellung des Prototyps zum Einsatz kam. Der Prototyp wurde für die Evaluation der Methode und der Architektur verwendet und in diesem Rahmen gegen die oben genannten Anforderungen getestet.

7.2 Architekturkomponenten

Die Architektur wird in die fünf Hauptkomponenten Modellierungskomponente, Überwachungsanwendung, Log-Agenten, Log-Filter und Prüfeinheit untergliedert. Abbildung 7.1 zeigt die einzelnen Komponenten im Überblick, die nachfolgend im Detail beschrieben werden.

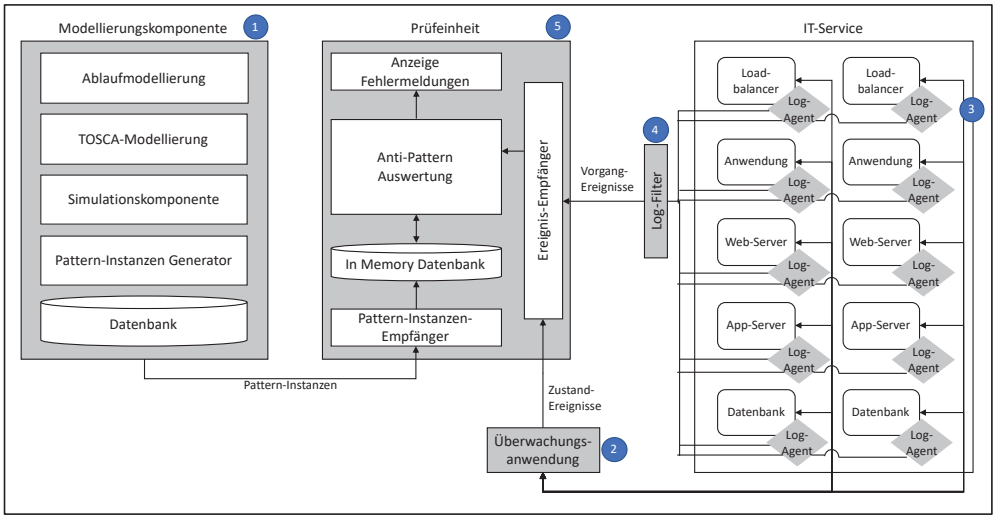


Abbildung 7.1: Architekturüberblick

7.2.1 Modellierungskomponente

Die Modellierungskomponente unterteilt sich weiter in eine Komponente zur graphischen Modellierung von Wartungsplänen und einer Komponente zur graphischen Modellierung von Service Templates in TOSCA. Die modellierten TOSCA Service Templates können bei den Wartungsplänen als Marken verwendet werden und die definierten Node Types den Stellen in einem Wartungsplan zugeordnet werden.

Um Pattern-Instanzen für die modellierten Wartungspläne zu erzeugen, wird eine Simulationskomponente benötigt, welche die Wartungspläne, wie in Kapitel 6.4.1 beschrieben, simuliert und das erzeugte Ereignis-Log in einer Datenbank speichert. Sobald die Simulation beendet ist, erzeugt der Pattern-Instanzen Generator aus diesem Ereignis-Log die Pattern-Instanzen für den Wartungsplan und sendet diese in einem standardisierten Format wie JSON oder XML an die Prüfeinheit.

7.2.2 Überwachungsanwendung

Die Überwachungskomponente ist eine Anwendung zur Überwachung des Status von IT-Service-Komponenten. Diese Anwendung prüft kontinuierlich, ob eine IT-Service-Komponente erreichbar ist oder nicht und sendet das Ergebnis in Form eines Zustand-Ereignisses an die Prüfeinheit. Das Zustand-Ereignis muss dabei in einem standardisierten Format an die Prüfeinheit übertragen werden, beispielsweise in JSON. Die Frequenz der Prüfung muss so eingestellt werden, dass der Zustand einer IT-Service-Komponente zwischen mindestens zwei Operationen ermittelt werden kann. Wenn beispielsweise ein Server gestoppt und wieder gestartet werden soll, dann muss die Überwachungsanwendung in der Lage sein den Status der IT-Service-Komponente nach dem Stoppen der Anwendung und vor dem Starten der Anwendung zu ermitteln. Für den in dieser Arbeit entwickelten Prototypen und die durchgeführten Evaluationen ist diese Architektur ausreichend, da alle Operationen auf den IT-Service-Komponenten manuell durchgeführt werden. Wenn Operationen auf IT-Service-Komponenten (teil-)automatisiert durchgeführt werden, wird eine solche Architektur jedoch zu einer sehr hohen Frequenz an Statusabfragen von der Überwachungsanwendung an die IT-Service-Komponenten führen. Der Grund ist, dass bei einer (teil-)automatisierten Wartung die Statusübergänge schneller erfolgen als bei einer manuellen Wartung und somit die Frequenz an Statusabfragen deutlich erhöht werden muss, um keine Statusübergänge zu verpassen.

Für die (teil-)automatisierte Wartung kann die Architektur dahingehend angepasst werden, dass eine Anwendung direkt auf den zu überwachenden IT-Service-Komponenten installiert

wird und Statusänderungen aktiv an die Prüfeinheit sendet. In diesem Fall kann die Frequenz der weiterhin notwendigen Statusabfragen von der Überwachungsanwendung an die IT-Service-Komponenten reduziert werden, um ungeplante Statusänderungen, z.B. bei einem Hardwareausfall, zu erkennen.

Für einen Überblick über einige am Markt bestehende Überwachungsanwendungen für IT-Service-Komponenten siehe [HeGS15].

7.2.3 Log-Agenten

Log-Agenten sind Anwendungen, die die Logs von IT-Service-Komponenten kontinuierlich überwachen. Wenn ein Log-Agent einen neuen Eintrag im Log einer IT-Service-Komponente erkennt, sendet es diesen an den zentralen Log-Filter. Damit der Log-Filter das erhaltene Ereignis einer IT-Service-Komponente zuordnen kann, muss der Log-Agent vor dem Versand eines Log-Eintrags diesen um die Werte st und app, die die IT-Service-Komponente eindeutig identifizieren, ergänzen.

Um Konfigurationsänderungen zu erkennen, kann es zudem notwendig sein, dass der Log-Agent diese Änderungen selbst loggen muss. Häufig werden Konfigurationsänderungen bei IT-Service-Komponenten direkt in einer separaten Konfigurationsdatei durchgeführt und erscheinen dadurch in keinem Log. Um auch diese Konfigurationsänderungen zu erfassen, muss der Log-Agent eine Kopie der Konfigurationsdatei bereithalten. Bei einer Änderung der Konfigurationsdatei kann diese anschließend mit der Kopie abgeglichen werden, um Änderungen identifizieren und anschließend loggen zu können. Für das Element $timestamp$ wird in diesem Fall der Zeitstempel der letzten Änderung der Konfigurationsdatei gesetzt.

7.2.4 Log-Filter

Im Log-Filter werden die von den Log-Agenten erhaltenen Ereignisse analysiert, und in das in Kapitel 5.3 beschriebene Format eines Vorgang-Ereignisses transformiert. Die Elemente $timestamp$, op, $prop$ und $value$ werden hierfür mittels regulären Ausdrücken aus dem erhaltenen Ereignis gelesen. Für den Wert des Elements op muss vor dem Versand an die Prüfeinheit eine Transformation durchgeführt werden, so dass ein Wert verwendet wird, der auch bei den *Operationen* im jeweiligen Node Type der IT-Service-Komponente definiert ist. Dazu muss vorab identifiziert werden, welcher Log-Eintrag welcher Operation entspricht. Wenn beispielsweise im Node Type einer IT-Service-Komponente die *Operation* „stop“ definiert ist, im Log der IT-Service-Komponente jedoch der Eintrag „shutting

down...“ verwendet wird, so muss dieser Wert vor dem Versand an die Prüfeinheit im Vorgang-Ereignis in „stop“ geändert werden.[1] Die Daten müssen in einem standardisierten Format wie JSON oder XML versendet werden.

7.2.5 Prüfeinheit

Die Prüfeinheit enthält Empfänger für die eintreffenden Vorgang-Ereignisse, Zustand-Ereignisse und die Pattern-Instanzen. Die Pattern-Instanzen, sowie die Vorgang-Ereignis-Historie und Zustand-Ereignis-Historie werden in einer In Memory Datenbank gespeichert, um eine schnelle Verarbeitung zu ermöglichen. Die Auswertung der Ereignisse auf die Anti-Pattern erfolgt nach dem Prinzip des Complex Event Processings, welches nachfolgend detailliert vorgestellt wird. Zusätzlich enthält die Prüfeinheit eine Komponente zur Anzeige von Fehlermeldungen.

7.3 Exkurs: Complex Event Processing

Complex Event Processing ist eine Softwaretechnologie, die von David Luckham [Luck01] für die kontinuierliche Online-Verarbeitung von Ereignissen entwickelt wurde. Ereignisse werden im Rahmen des Complex Event Processings beschrieben als etwas, das passiert oder wovon angenommen wird, dass es passiert [LSAB11]. Ein Beispiel für ein Ereignis ist die Landung eines Flugzeugs oder die Überweisung eines Geldbetrags. Ein Ereignis wird im Rahmen der elektronischen Datenverarbeitung durch ein Ereignis-Objekt repräsentiert [LSAB11].[2] Ein komplexes Ereignis ist wiederum ein Ereignis, das mehrere Ereignisse repräsentiert oder zusammenfasst [LSAB11]. Beispielhaft sei der Kauf einer Immobilie genannt, der unter anderem mehrere Überweisungen von Geldbeträgen beinhalten kann.

[1] Bei einer hohen Anzahl solcher Transformationen kann die Erstellung eine Ontologie von Nutzen sein. Im Rahmen des Prototyps wurde eine solche nicht erstellt, da die Zahl an Transformationen im in Kapitel 8.1 beschriebenen Versuchsaufbau überschaubar war. Für eine Konzeption einer Onotologie speziell für Petri-Netze sei auf [Kosc07] verwiesen.

[2] Luckham et al. [LSAB11] weisen darauf hin, dass der Begriff Ereignis überladen ist und häufig sowohl für ein Ereignis-Objekt, als auch für ein Ereignis in Form, dass etwas passiert, verwendet wird. Da beim Complex Event Processing mit Ereignis-Objekten gearbeitet wird, müsste die Begriffswelt entsprechend angepasst werden, z.B. Complex Event Object Processing. Da dies jedoch das Leseverständnis erschwert, wird bewusst der Begriff Ereignis auch für ein Ereignis-Objekt verwendet. Diese Aufweichung des Begriffs wird bewusst in Kauf genommen, da sich der eigentliche Sinn in der Regel durch den Kontext, in dem der Begriff Ereignis verwendet wird, ergibt.

Zur Erkennung von Zusammenhängen zwischen Ereignissen werden auftretende Ereignisse auf Ereignismuster untersucht. Ein Beispiel für ein Ereignismuster ist das Auftreten von Überweisungen von einer bestimmten Person. Ereignismuster können unterteilt werden in einfache Ereignismuster zur Erkennung von einzelnen Ereignissen und komplexen Ereignismustern, die mehrere Ereignisse in Beziehung zueinander setzen [Luck01, S.164]. Ereignismuster sind beim Complex Event Processing in Ereignisregeln eingebettet, die spezifizieren was passieren soll, wenn ein Ereignismuster erkannt wird. Ereignisregeln repräsentieren somit explizit und deklarativ das Wissen über die Ereignisverarbeitung [BrDu15, S.11] und sind aus folgenden zwei Teilen aufgebaut:

- Bedingungsteil
  Der Bedingungsteil enthält ein oder mehrere Ereignismuster. Dieser Bedingungsteil wird gegen alle zu untersuchenden Ereignisse abgeglichen. Sobald ein Ereignismuster in den untersuchten Ereignissen entdeckt wird, wird der Aktionsteil der Ereignisregel ausgeführt.
- Aktionsteil
  Der Aktionsteil der Ereignisregel führt eine Aktion aus, sobald der Bedingungsteil erfüllt ist. Eine solche Aktion ist beispielsweise die Erstellung eines neuen Ereignisses.

Die Kernkomponente zur Verarbeitung von Ereignissen ist der Event Processing Agent. Der Event Processing Agent kann Ereignisse in einem Ereignisstrom verarbeiten und besteht aus folgenden drei Elementen [BrDu15, S.13f.]:

- Ereignismodell
  Das Ereignismodell definiert den Typ der erlaubten Ereignisse und die Datenattribute der erlaubten Ereignistypen.
- Ereignisregeln
  Unter Berücksichtigung der Ereignistypen, die im Ereignismodell festgelegt werden, werden Ereignisregeln definiert. Wenn die Daten in einem Ereignis-Objekt nicht für die Verarbeitung ausreichen, können diese mit Kontextwissen angereichert werden. Zur Nutzung von Kontextwissen können beispielsweise Datenbanken verwendet werden.
- Event Processing Engine
  Die Event Processing Engine ist ein Regelinterpreter und führt die eigentliche Mustererkennung durch. Dazu werden die Ereignisse kontinuierlich auf Muster, die im Bedingungsteil einer Ereignisregel definiert sind, abgeglichen. Damit auch bereits in der Vergangenheit erfasste Ereignisse für die Mustererkennung genutzt werden

können, werden diese im Arbeitsspeicher gespeichert. Aufgrund der typischerweise sehr großen Menge an zu verarbeitenden Ereignissen, können die im Arbeitsspeicher zu speichernden Ereignisse auf bestimmte Zeit- oder Mengenintervalle eingeschränkt werden.

Event Processing Agents ermöglichen die Realisierung eines Regelinterpreters, mit dem wenige abgegrenzte Regeln überprüft werden können. Sollen sehr viele aufeinander aufbauende Regeln abgebildet werden, können mehrere Event Processing Agents zu einem Event Processing Network verknüpft werden [Luck01, S.208]. Die Verknüpfung zwischen den Event Processing Agents erfolgt durch Ereigniskanäle, die für die Übertragung von Ereignissen genutzt werden [LSAB11].

Da die Beschreibung von Ereignismustern ein wesentlicher Bestandteil des Complex Event Processings ist, werden nachfolgend die wesentlichen Sprachkonzepte zur Beschreibung von Ereignismustern vorgestellt.

7.3.1 Sprachkonzepte

Generell können die vier Sprachkonstrukte Operatoren, Fenster, Aggregationen und Kontextbedingungen bei der Spezifikation von Ereignismustern unterschieden werden [BrDu15].

Operatoren werden unterteilt in die Operatoren ∧ , ∨, → und ¬ [Luck01, S.116]. ∧ und ∨ definieren, welche Ereignisse eines Typs unabhängig von einer Reihenfolge vorliegen müssen. $A \vee B$ bedeutet beispielsweise, dass entweder ein Ereignis des Typs A oder ein Ereignis des Typs B vorliegen muss. $A \wedge B$ bedeutet, dass jeweils ein Ereignis des Typs A und des Typs B auftreten muss. → beschreibt einen kausalen Zusammenhang, d.h. ein Ereignis eines Typs muss nach dem Auftreten eines anderen Ereignisses auftreten. Beispielsweise bedeutet $A \rightarrow B$, dass ein Ereignis des Typs B nach einem Ereignis des Typs A auftreten muss. Mit dem Negationsoperator ¬ kann angegeben werden, dass ein Ereignis eines bestimmten Typs nicht auftreten darf. Der Negationsoperator kann nur auf einen Ausschnitt eines Ereignisstroms angewendet werden, da ein Nicht-Auftreten bei einem nicht aufhörendem Ereignisstrom nicht überprüft werden kann [EcBr09]. Ein solcher Ausschnitt eines Ereignisstroms wird als Fenster bezeichnet.

Ein Fenster schränkt den betrachteten Ereignisstrom auf eine endliche Menge an Ereignissen ein. Dabei kann unterschieden werden in Zeitfenster und Längenfenster [BrDu15]. Bei einem Längenfenster wird nur eine bestimmte Anzahl der letzten Ereignisse im Ereignisstrom betrachtet. Längenfenster arbeiten nach dem Prinzip first-in-first-out. Bei

Zeitfenstern wird ein Zeitraum definiert, in dem Ereignisse gespeichert werden. Wenn beispielsweise ein Zeitfenster von fünf Minuten spezifiziert wird, werden nur die Ereignisse, die in den letzten fünf Minuten aufgetreten sind, gespeichert. Vergehen mehr als fünf Minuten seit dem Auftreten eines Ereignisses, wird dieses aus dem Zeitfenster entfernt.

Ein weiteres Sprachkonstrukt, das durch die Nutzung von Zeitfenstern ermöglicht wird, ist die Aggregation von Attributwerten der Ereignisse in einem Zeitfenster [BrDu15]. Typische Aggregationsfunktionen sind

- $sum()$ für die Bildung von Summen,
- $avg()$ für die Bildung von Durchschnittswerten, sowie
- $min()$ und $max()$ zur Identifikation des kleinsten bzw. größten Attributwertes.

Um Anforderungen an Attributwerte definieren zu können, werden Kontextbedingungen eingesetzt. Dazu werden Aliasnamen und der „.“-Operator für den Zugriff auf ein Attribut eines Ereignisses verwendet. Aliasnamen dienen der Unterscheidung von Ereignis-Instanzen, in dem diese mit dem Schlüsselwort „AS“ ein Name zugewiesen werden kann. Attributwerte können durch typische Operatoren miteinander verglichen werden. Bei numerischen Attributwerten können dies beispielsweise die Operatoren $<, >, +, -, *$ usw. sein [BrDu15].

Nachfolgend wird anhand der Open-Source-CEP-Engine „WSO2 Complex Event Processor“[3] und der verwendeten Ereignisverarbeitungssprache SiddhiQL eine konkrete Realisierungsplattform beschrieben. Diese wird auch für die Umsetzung des Prototyps verwendet. Für eine Übersicht bestehender CEP-Engines siehe [Vinc16]. Der Complex Event Processor von WSO2 wurde ausgewählt, da dieser aufgrund der Apache[4]-Lizenzierung frei nutzbar ist und zudem eine graphische Benutzeroberfläche bereitstellt.

7.3.2 WSO2 Complex Event Processor und SiddhiQL

Die Architektur des WSO2 Complex Event Processors unterteilt sich in die Komponenten Event Receivers, Event Streams, Event Processors und Even Publishers. Event Receivers dienen dem Empfang von Ereignissen[5]. Es werden mehrere Adapter bereitgestellt, um verschiedene Quellsysteme bzw. Formate wie beispielsweise Email, SOAP-Nachrichten oder Einträge in einer Textdatei anbinden zu können.

[3] http://wso2.com/products/complex-event-processor/
[4] https://www.apache.org/licenses/LICENSE-2.0
[5] Die Beschreibungen, Abbildungen und Quelltexte in diesem Unterkapitel sind der Dokumentation von WSO2 übernommen [Wso17]. Für eine weiterführende Dokumentation sei auf diese verwiesen.

Event Streams enthalten eine Menge von Ereignissen eines bestimmten Ereignistyps. Ein Event Processor ist ein Complex Event Processing Agent, der die eigentliche Mustererkennung und Verarbeitung von Ereignissen durchführt. Die Event Processing Engine wird Siddhi Core genannt. Event Publishers dienen der Veröffentlichung von Ereignissen in verschiedenen Zielsystemen, die über Adapter direkt angebunden werden können. Bereitgestellte Adapter sind z.B. http-, SAOP- oder Datenbankadapter.

Abbildung 7.2 stellt den Aufbau des WSO2 Complex Event Processors graphisch dar.

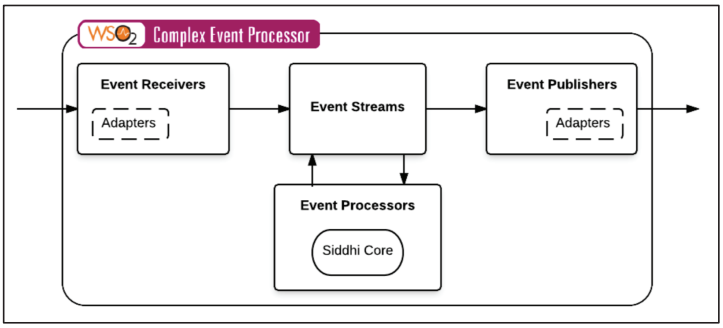


Abbildung 7.2: Architektur des WSO2 Complex Event Processors

Zur Beschreibung von Ereignisregeln wird die Sprache SiddhiQL verwendet, welche nachfolgend in ihren Grundzügen beschrieben wird. Quelltext 7.1 zeigt den grundlegenden Aufbau einer Ereignisregel, welcher aus einer **from**-Klausel, einer **select**-Klausel und einer **insert into**-Klausel besteht. In der **from**-Klausel wird angegeben welcher Event Stream ausgewertet wird. Die **select**-Klausel ermöglicht die Manipulation von Ereignissen. Die **insert into**-Klausel gibt an in welchen Event Stream Ereignisse hinzugefügt werden.

```
1  from <input stream name>
2  select <attribute name>, <attribute name>, ...
3  insert into <output stream name>
```

Quelltext 7.1: Aufbau einer SiddhiQL-Ereignisregel

Filter ermöglichen die Auswahl bestimmter Ereignisse aus einem Event Stream und werden wie in Quelltext 7.2 gezeigt angegeben.

```
1  from <input stream name> [<filter condition>]
```

Quelltext 7.2: Filter

Fenster werden in einer **from**-Klausel durch den Zusatz **#window.<window name> (<parameter>)** angegeben. Es werden verschiedene Längen- und Zeitfenster bereitgestellt, z.B. dass eine vorab definierte Menge von Ereignissen gespeichert werden soll, oder nur das erste Ereignis eines Typs gespeichert werden wird. Quelltext 8 zeigt die Nutzung eines Fensters in einer **from**-Klausel.

```
1  from <input stream name> [<filter condition>]#window.<window name> (<parameter>, <parameter>, …)
```

Quelltext 7.3: Fenster

In der **insert into**-Klausel können drei verschiedene Arten von Ereignissen ausgegeben werden:

- **current events**
- **expired events**
- **all events**

Bei der Verwendung von **current events** werden nur die Ereignisse ausgegeben, die in einem Fenster gespeichert sind. Bei der Verwendung von **expired events** werden nur Ereignisse ausgegeben, die aus einem Fenster entfernt werden. Bei der Verwendung von **all events** werden sowohl sich in einem Fenster befindende Ereignisse, als auch aus einem Fenster ausscheidende Ereignisse ausgegeben. Quelltext 7.4 zeigt wie die Ereigniskategorie in der **insert into**-Klausel angegeben wird.

```
1  insert current events into <output stream name>
```

Quelltext 7.4: insert into Ereigniskategorien

Für die Manipulation von Ereignissen werden in der **select**-Klausel verschiedene Möglichkeiten zur Verfügung gestellt. Die Umbenennung von Attributen erfolgt beispielsweise durch das Wort **as**. Außerdem werden die in Kapitel 7.3.1 vorgestellten relationalen Operatoren durch die Angabe von **AND**, **OR** bzw. **->,** sowie verschiedene Aggregationsfunktionen unterstützt. Zusätzlich werden Vergleichsoperatoren wie **<**, **>**, **+**, etc. unterstützt. Quelltext 7.5 zeigt eine beispielhafte Ereignisregel, in der die Raumtemperatur eines Raums von Celsius in Fahrenheit umgerechnet wird. Zusätzlich wird in einem Attribut angegeben, ob es sich bei diesem Raum um einen Server-Raum handelt. Dazu werden die Ereignisse im Event Stream TEMPSTREAM ausgewertet und in den Event Stream ROOMTEMPSTREAM eingefügt. In der **select**-Klausel wird der Wert des Attributs TEMP von Celsius in Fahrenheit umgerechnet und im neu erstellten Ereignis wiederum als Attribut TEMP gespeichert.

Zusätzlich wird das Attribut ROOMNO übergeben, sowie zwei neue Attribute SCALE und ISSERVERROOM erstellt. Das Attribut SCALE enthält standardmäßig den Wert F. Das Attribut ISSERVERROOM enthält den Wert TRUE, wenn der Wert des Attributs ROOMNO größer gleich 100 und kleiner 110 ist.

```
1  from TempStream
2  select temp * 9/5 + 32 as temp, roomNo, 'F' as scale, roomNo >= 100 and roomNo <
   110 as isServerRoom
3  insert into RoomTempStream;
```

Quelltext 7.5: Beispielhafte SiddhiQL-Ereignisregel

Zur Nutzung von Kontextwissen können **Event Tables** verwendet werden. Ein **Event Table** ist eine Tabelle, in der Ereignisse gespeichert werden. **Event Tables** können im Arbeitsspeicher in Form einer In Memory Datenbank gespeichert werden. Es können jedoch auch klassische relationale Datenbanksysteme angebunden werden. Durch die Nutzung des **IN**-Operators kann geprüft werden, ob Attribute eines Ereignisses in einer **Event Table** enthalten sind. Der **JOIN**-Operator ermöglicht den Verbund von Ereignissen in einem **Event Stream** mit Ereignissen aus einer **Event Table** oder einem Fenster. Mit der Angabe von **on <join condition>** können Bedingungen für die Verbundoperation definiert werden.

Nachdem die Sprache SiddhiQL in ihren Grundzügen vorgestellt wurde, wird nachfolgend beschrieben, wie Anti-Pattern in SiddhiQL definiert werden können.

7.3.3 Anti-Pattern als SiddhiQL-Anfrage

In diesem Unterkapitel wird anhand der beiden Anti-Pattern NACHFOLGER und DIREKTER NACHFOLGER exemplarisch beschrieben, wie diese als SiddhiQL-Ereignisregeln formuliert werden können. Die Umsetzung der weiteren Anti-Pattern in SiddhiQL-Ereignisregeln kann Anhang A entnommen werden. Bei allen Ereignisregeln wird der Event Stream VORGANG_EREIGNISSE_STREAM ausgewertet. Dieser Event Stream enthält alle Vorgang-Ereignisse der überwachten IT-Service-Komponenten. Pattern-Instanzen werden in einer In Memory Event Table gespeichert. Die Event Tables sind entsprechend den Anti-Pattern benannt, z.B. NACHFOLGER. Fehlermeldungen werden als Ereignisse in den Event Stream FEHLERMELDUNGEN_STREAM eingegeben. Alle Vorgang-Ereignisse werden zudem in der In Memory Event Table VORGANGEREIGNISHISTORIE gespeichert.

Quelltext 7.6 zeigt das Anti-Pattern NACHFOLGER als Ereignisregel. In dieser Ereignisregel wird in Zeile 1 zunächst überprüft, ob ein Vorgang-Ereignis einer Aktivität a_{akt} einer Pattern-Instanz zugeordnet werden kann und somit für die weitere Überprüfung relevant ist. Dazu findet ein Abgleich mit den Attributen $a_{akt}.app$, $a_{akt}.op$ und $a_{akt}.prop$ statt. Relevante Vorgang-Ereignisse werden in Zeile 3 mit den Pattern-instanzen in der Event Table NACHFOLGER verbunden. Dadurch werden neue Ereignisse erzeugt, die zusätzlich die Attribute der Aktivitäten a_{nach} und a_{entf} enthalten. In Zeile 7 wird überprüft, ob eines der nachfolgenden Vorgang-Ereignisse einer Aktivität a_{entf} entspricht (dazwischen können mehrere Ereignisse auftreten, z.B. erfüllt eine Ereignisfolge a, b, c, d, e das Muster $a -> e$). Wenn eines der nachfolgenden Vorgang-Ereignisse a_{entf} entspricht, wird in Zeile 10 geprüft, ob die Aktivität a_{nach} bereits in der Vorgang-Ereignis-Historie enthalten ist. Ist dies nicht der Fall, wird eine Fehlermeldung erzeugt und in Form eines neuen Ereignisses (Zeile 11) in den Event Stream FEHLERMELDUNGEN_STREAM eingefügt (Zeile 12).

```
1  from eingehende_VorgangEreignisse [(app == NACHFOLGER.appakt and op ==
   NACHFOLGER.opakt and prop == NACHFOLGER.propakt and value == NACHFOL-
   GER.valueakt) in NACHFOLGER]
2  insert into #temp;
3  from #temp as tmp join NACHFOLGER as nachf
4  on tmp.app == nachf.appakt and tmp.op == nachf.opakt and tmp.prop ==
   nachf.propakt and tmp.value == nachf.valueakt
5  select tmp.timestamp, nachf.appakt, nachf.opakt, nachf.propakt, nachf.valueakt,
   nachf.appnach, nachf.opnach, nachf.propnach, nachf.valuenach, nachf.appentf,
   nachf.opentf, nachf.propentf, nachf.valueentf
6  insert into #temp1;
7  from every e1=#temp1 -> e2=eingehende_VorgangEreignisse [e1.appentf ==
   e2.app and e1.opentf == e2.op and e1.propentf == e2.prop and e1.valueentf ==
   e2.value and e1.timestamp != e2.timestamp]
8  select e1.timestamp, e1.appakt, e1.opakt, e1.propakt, e1.valueakt, e1.appnach,
   e1.opnach, e1.propnach, e1.valuenach, e2.timestamp as timestampentf
9  insert into #temp2;
10 from #temp2 [not((appnach == VorgangEreignisHistorie.app and opnach == Vor-
   gangEreignisHistorie.op and propnach == VorgangEreignisHistorie.prop and valu-
   enach == VorgangEreignisHistorie.value and time:timestampInMilliseconds(ti-
   mestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <
   time:timestampInMilliseconds(VorgangEreignisHistorie.timestamp, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") and time:timestampInMilliseconds(timestampentf, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'") > time:timestampInMilliseconds(VorgangEreignisHis-
   torie.timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")) in VorgangEreignisHistorie)]
11 select str:concat("Die Aktivitaet ", appnach, ", ", opnach, ", ", propnach, ",", valu-
   enach, " wurde nicht nach der Aktivitaet ", appakt, " ,", opakt, ", ", propakt, ", ", va-
   lueakt," durchgefuehrt.") as Fehlermeldung
12 insert into Fehlermeldungen_Stream;
```

Quelltext 7.6: NACHFOLGER als SiddhiQL-Ereignisregel

Quelltext 7.7 zeigt das Anti-Pattern DIREKTER NACHFOLGER als Ereignisregel. In dieser Ereignisregel werden alle direkt nacheinander folgenden Vorgang-Ereignisse $e1$ und $e2$, die demselben Service Template angehören, ausgewertet[6]. Werden zwei direkt nacheinander

[6] Im Gegensatz zum „−>“-Operator, bedeutet der „,“-Operator, dass zwischen zwei gesuchten Ereignissen kein drittes Ereignis auftreten darf, z.B. erfüllt eine Ereignisfolge a, b, c, d, e das Muster a, e nicht.

folgende Vorgang-Ereignisse desselben Service Templates erkannt, wird ein neues Ereignis erzeugt, welches jeweils die Attribute beider Vorgang-Ereignisse enthält (Zeile 1-3). Anschließend wird überprüft, ob die Attribute app, op und $prop$ der beiden Ereignisse einem Aktivitätenpaar einer Pattern-Instanz des Typs DIREKTER NACHFOLGER zugeordnet werden kann (Zeile 4). Ist dies nicht der Fall, wird eine Verbundoperation mit den Pattern-Instanzen vorgenommen (Zeile 6-7). Wenn eine Aktivität gefunden wurde, die dem ursprünglichen Ereignis $e1$ entspricht, wird eine Fehlermeldung ausgegeben (Zeile 8-9). Ergibt die Verbundoperation eine leere Menge, wird keine Fehlermeldung ausgegeben, da dies bedeutet, dass auf das Ereignis $e1$ keine Aktivität direkt folgen muss.

```
1  from e1=eingehende_VorgangEreignisse , e2=eingehende_VorgangEreignisse[e1.st
   == e2.st]
2  select e1.app as appakt, e1.op as opakt, e1.prop as propakt, e1.value as valueakt,
   e2.app as istappnachf, e2.op as istopnachf, e2.prop as istpropnachf, e2.value as
   istvaluenachf
3  insert into #temp;
4  from #temp [not((DirektePaare.appvor == appakt and DirektePaare.opvor == opakt
   and DirektePaare.propvor == propakt and DirektePaare.valuevor == valueakt and
   DirektePaare.appnachf == istappnachf and DirektePaare.opnachf == istopnachf and
   DirektePaare.propnachf == istpropnachf and DirektePaare.valuenachf == istvalu-
   enachf) IN DirektePaare)]
5  insert into #temp1;
6  from #temp1 as tmp join DirektePaare as nachf
7  on tmp.appakt == nachf.appvor and tmp.opakt == nachf.opvor and tmp.propakt ==
   nachf.propvor and tmp.valueakt == nachf.valuevor
8  select str:concat("Die Aktivitaet ", nachf.appnachf, ", ", nachf.opnachf, ", ",
   nachf.propnachf, ", ", nachf.valuenachf," wurde nicht direkt nach der Aktivitaet ",
   tmp.appakt, " ,", tmp.opakt, ", ", tmp.propakt, ", "," durchgefuehrt.") as Fehlermel-
   dung
9  insert into Fehlermeldungen_Stream;
```

Quelltext 7.7: DIREKTER NACHFOLGER als SiddhiQL-Ereignisregel

Nachdem anhand der beiden Anti-Pattern NACHFOLGER und DIREKTER NACHFOLGER beschrieben wurde, wie Anti-Pattern als SiddhiQL-Ereignisregeln definiert werden können, wird nachfolgend eine prototypische Implementierung der in Kapitel 7.2 beschriebenen Architekturkomponenten vorgestellt. Die in diesem Kapitel und in Anhang A beschriebenen

SiddhiQL-Ereignisregeln wurden im Rahmen der prototypischen Implementierung eingesetzt und in den in Kapitel 8 beschriebenen Evaluationsexperimenten auf ihre Korrektheit überprüft.

7.4 Prototypische Implementierung

Die Modellierungskomponente wird auf Basis des Modellierungswerkzeugs Horus[7] entwickelt, welches auch in der betrieblichen Praxis zum Einsatz kommt [SVFO16]. Horus ermöglicht die Modellierung sowohl von Abläufen als auch von weiteren Perspektiven auf Abläufe, wie beispielsweise Risiken, Ressourcen und im Ablauf verarbeitete Objekte. Für die Modellierung von Abläufen werden XML-Netze eingesetzt. Zur Modellierung von TOSCA-Service Templates wurde ein eigener Editor in Horus implementiert. In diesem Editor können TOSCA-Service Templates modelliert und anschließend in einem Ablaufmodell als Marke zugeordnet werden. Zur Simulation von Abläufen enthält Horus eine Simulationskomponente. Diese Simulationskomponente ist noch zur Simulation von Wartungsplänen so anzupassen, dass die in Form von Marken hinterlegten Service Templates bei der Schaltung von Transitionen entsprechend berücksichtigt werden. In diesem Rahmen ist zudem noch der Generator zur Instanziierung der Pattern-Instanzen, wie in Kapitel 6.4 beschrieben, zu entwickeln.

Die Prüfeinheit wurde mit dem Complex Event Processor von WSO2 umgesetzt. Dazu wurden Event Receiver und Event Streams für Vorgang- und Zustand-Ereignisse, sowie für die einzelnen Typen von Pattern-Instanzen erstellt. Die Anti-Pattern wurde alle in einem „Execution Plan“ definiert, in dem die Auswertung der Event-Streams durch den Event Processor erfolgt. Zudem werden alle Pattern-Instanzen im Rahmen der Verarbeitung im „Execution Plan“ in In Memory Event tables gespeichert. Die Fehlermeldungen werden in der Konsole des Complex Event Processors ausgegeben. Abbildung 7.3 zeigt einen Screenshot des Prüfeinheit-Prototyps, in dem der Datenfluss zwischen den einzelnen Complex Event Processing Komponenten abgebildet ist.

[7] http://www.horus.biz/de/produkte/business-modeler/

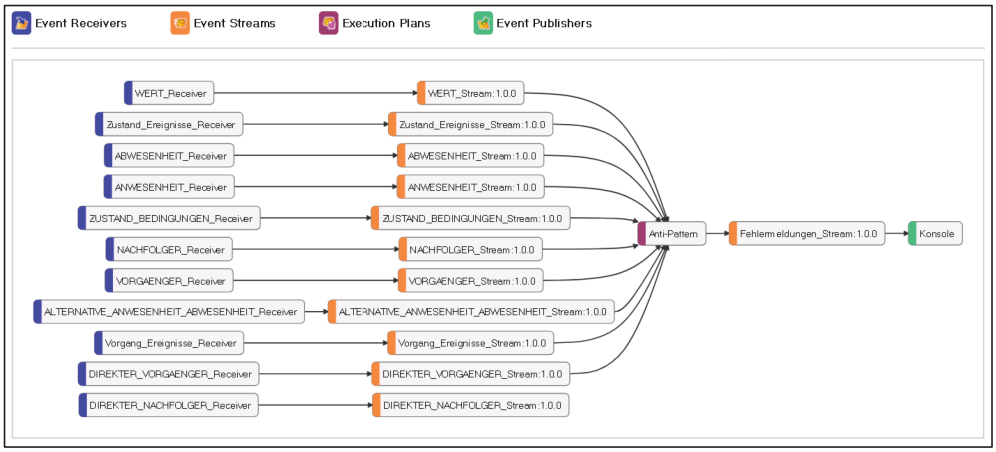


Abbildung 7.3: Complex Event Processing Screenshot

Für die Umsetzung des Log-Filters und der Log-Agenten wurden die frei verfügbaren Open Source-Anwendungen Logstash[8] und Filebeat[9] eingesetzt, die speziell für den Zweck der Verarbeitung von Log-Daten entwickelt wurden. Filebeat wird als Log-Agent direkt auf der zu überwachenden IT-Service-Komponente installiert und liest die Log-Files der IT-Service-Komponenten aus. Jeder neue Log-Eintrag wird an Logstash gesendet. Logstash übernimmt die Aufgaben des Logfilters und transformiert die erhaltenen Log-Einträge von Filebeats, um diese anschließend an den Complex Event Processor von WSO2 zu senden. Das Logging von Konfigurationsänderungen wurde durch die eigenentwickelte Java-Anwendung „Configlogger“ vorgenommen. Die Überwachungsanwendung wurde mit Metricbeat[10] umgesetzt. Metricbeat sendet in einstellbaren Zeitintervallen eine Anfrage an die zu überwachenden IT-Service-Komponenten und leitet die Antwort, ob die IT-Service-Komponente erreichbar ist oder nicht, an Logstash weiter. In Logstash wird die Antwort in das Format eines Zustand-Ereignisses transformiert und anschließend an den Complex Event Processor gesendet.

Um die Pattern-Instanzen in WSO2 einzulesen, können die Pattern-Instanzen in einer relationalen Datenbank gespeichert werden, die sowohl von WSO2 als auch von Horus verwendet wird. Alternativ können statt einer direkten Anbindung von Horus an WSO2 mithilfe einer Datenbank auch die Pattern-Instanzen als Text-Dateien abgelegt werden und anschließend von Filebeat eingelesen und an Logstash gesendet werden. Dies wurde im

[8] https://www.elastic.co/products/logstash
[9] https://www.elastic.co/de/products/beats/filebeat
[10] https://www.elastic.co/guide/en/beats/metricbeat/index.html

Rahmen des Prototyps umgesetzt, wodurch es möglich ist, Pattern-Instanzen zu Evaluationszwecken manuell zu erstellen und in Form von Text-Dateien zu speichern.

Mithilfe dieses Prototyps wurde die in dieser Arbeit vorgestellte Methode zur Fehlererkennung evaluiert. Der Versuchsaufbau und die Evaluationsergebnisse werden im nachfolgenden Kapitel beschrieben.

7.5 Zusammenfassung des Kapitels

In diesem Kapitel wurde eine Architektur zur IT-gestützten Anwendung der in dieser Arbeit vorgestellten Methode konzipiert. Dazu wurden zunächst Anforderungen an die Architektur gestellt, die aus der Motivation zur Entwicklung der Methode abgeleitet wurden. Die Architektur kann unterteilt werden in Komponenten zur kontinuierlichen Datenextraktion und Datentransformation der Log-Informationen der zu wartenden IT-Service-Komponenten, sowie einer Komponente zur Analyse und Auswertung dieser Datenströme. Die Komponente zur Analyse und Auswertung der Daten beruht auf einer Complex Event Processing Engine, die es erlaubt Datenströme kontinuierlich gegen die in Kapitel 5.4 vorgestellten Anti-Pattern zu überprüfen. Um die Anti-Pattern in einer Complex Event Processing Engine umzusetzen, müssen diese in einer Event Processing Language beschrieben werden, die von der entsprechenden Complex Event Processing Engine unterstützt wird. Beispielhaft wurden dazu die beiden Anti-Pattern NACHFOLGER und DIREKTER NACHFOLGER in der Event Processing Language SiddhiQL beschrieben. Diese Sprache wurde auch für die prototypische Umsetzung der Architektur mit der Complex Event Processing Engine von WSO2 verwendet. Das Kapitel endet mit einer Beschreibung des entwickelten Prototyps, der für die Evaluation der Methode eingesetzt wurde. Die durchgeführte Evaluation und die Evaluationsergebnisse werden in nachfolgendem Kapitel präsentiert.

8 Evaluation

Nachdem im vorherigen Kapitel die Architektur und ein Prototyp zur IT-gestützten Umsetzung der Methode beschrieben wurden, werden in diesem Kapitel die Evaluationsergebnisse der Methode vorgestellt. Zunächst wird der Versuchsaufbau dokumentiert, d.h. der Aufbau der IT-Service-Komponenten, die für die Evaluation verwendet werden. Anschließend werden die durchgeführten Wartungen und injizierten Fehler beschrieben. Abschließend werden die Evaluationsergebnisse präsentiert.

8.1 Versuchsaufbau

Zur Evaluation der Methode wird eine Web-Anwendung genutzt. Web-Anwendungen sind in der betrieblichen Praxis zum einen sehr weit verbreitet. Zum anderen lassen sich damit typische Wartungsfehler, wie sie in Kapitel 3.3 beschrieben wurden, z.B. dass vergessen wird einen Server zu starten und dadurch eine Störung der gesamten Web-Anwendung verursacht wird, leicht simulieren. Als Web-Anwendung wird das Customer Relationship Management System SugarCRM[1] (Version 6.5.24) verwendet, welches Open-Source ist und als TOSCA Service Template verfügbar ist[2]. Um eine Komplexität zu erreichen, wie sie auch in der betrieblichen Praxis vorkommen kann, werden die einzelnen benötigten Komponenten des Systems, d.h. das Applikationstier, das Datenbanktier und der Loadbalancer auf separaten Servern installiert und redundant ausgelegt.

Die Versuchsumgebung wird auf insgesamt acht Servern aufgebaut. Als Server werden Server des Typs EC2[3] von Amazon Web Services[4] mit dem Betriebssystem Windows Server 2008 R2 eingesetzt. Auf zwei EC2-Servern wird die Applikation SugarCRM installiert. Da die Applikation eine Webanwendung ist, werden auf beiden EC2-Servern zusätzlich je ein Apache Webserver[5] (Version 2.4.29) sowie PHP 5 installiert. Zur Verteilung der Anfragen an die Applikation wird der Webserver nginx[6] (1.12.2) eingesetzt. Der Webserver fungiert in diesem Szenario als Loadbalancer und ist auf einem eigenen EC2-Server installiert. Der Loadbalancer verteilt Anfragen gleichmäßig auf die beiden EC2-Server, auf denen die

[1] https://www.sugarcrm.com
[2] https://www.oasis-open.org/committees/download.php/50158/SugarCRM-Interop-20130803.zip
[3] https://aws.amazon.com/de/ec2
[4] https://aws.amazon.com
[5] https://httpd.apache.org
[6] http://nginx.org

Applikation installiert ist. Die Applikation benötigt zur Speicherung von Daten eine Datenbank. Als Datenbankserver wird ein MySQL-Datenbankserver[7] (Version 5.7.21) auf einem eigenen EC2-Server installiert. Beide Applikations-Installationen verbinden sich zu derselben Datenbank auf dem Datenbankserver. Damit Konfigurationsänderungen erkannt werden können, wird auf allen EC2-Servern das eigenentwickelte Programm Configlogger installiert. Zum Versand von Vorgang-Ereignissen an die Prüfeinheit wird außerdem die Anwendung Filebeat (Version 6.1.2) installiert. Zusätzlich werden zwei Applikations-Instanzen mit exakt derselben Konfiguration als Backup-Lösung aufgebaut, die bei Bedarf gestartet werden können, standardmäßig jedoch gestoppt sind. Zudem wird ein Datenbankserver in exakt derselben Konfiguration erstellt, der ebenfalls als Backup-Lösung dient. Abbildung 8.1 verdeutlicht den Aufbau des gesamten Customer Relationship Management Systems. Auf dem achten EC2-Server wird der Complex Event Processor von WSO2 (Version 4.2.0) installiert. Außerdem werden Logstash (Version 5.2.0) und Metricbeat (Version 6.2.1) installiert.

[7] https://www.mysql.com/de

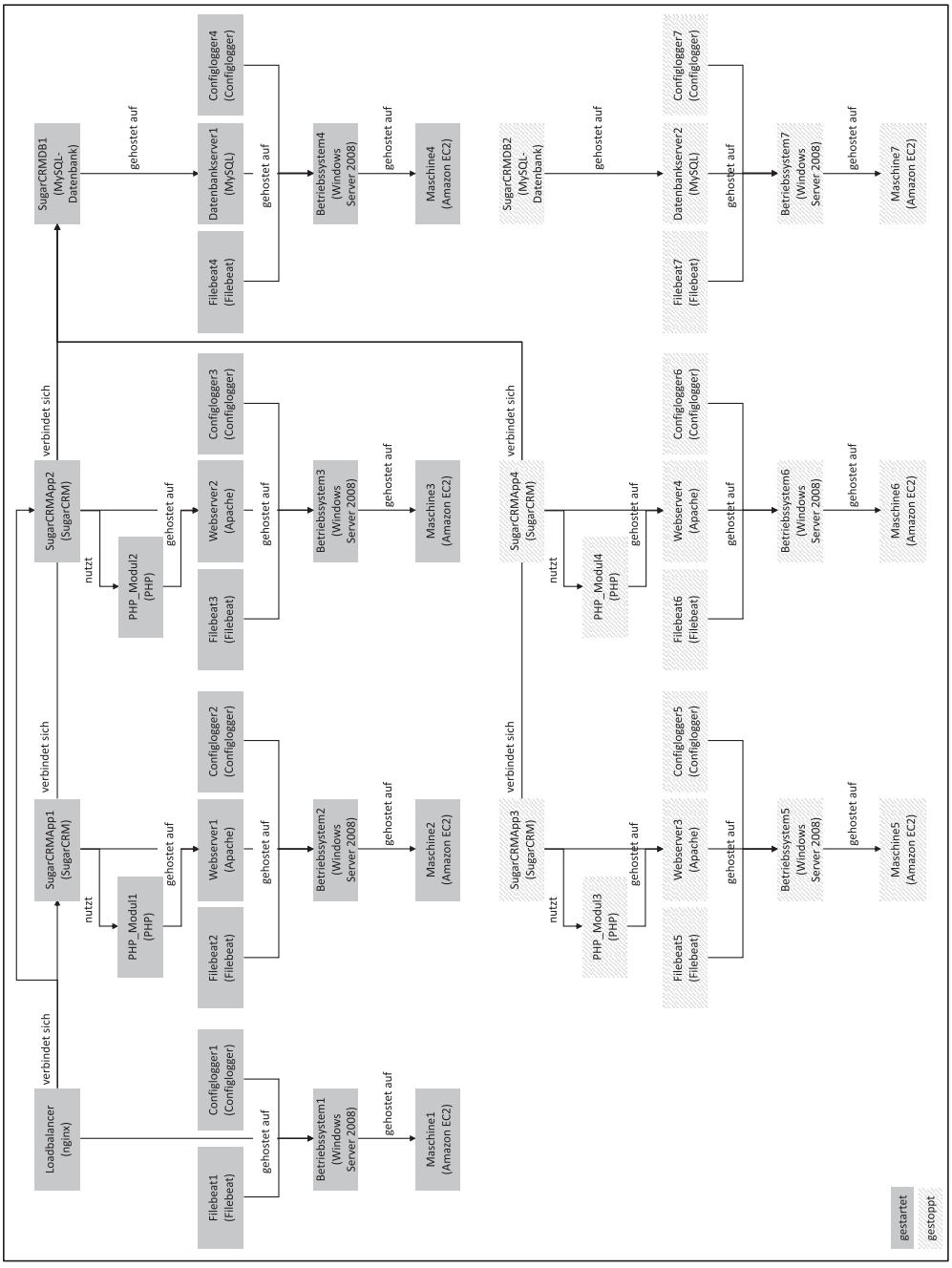


Abbildung 8.1: Versuchsaufbau

Diese Versuchsumgebung wird für verschiedene Experimente verwendet, um die in dieser Arbeit vorgestellte Methode zu evaluieren. Alternativ zu dieser Versuchsumgebung können auch andere Versuchsumgebungen genutzt werden, beispielsweise auf Basis des aktuellen Trends der Microservice-Architektur. In einer Microservice-Architektur werden im Vergleich zu einer monolitischen Architektur, wie es in diesem Versuchsaufbau verwendet wird, nicht die gesamte monolitische Anwendung auf mehrere Server repliziert. Stattdessen wird die monolitische Anwendung in einzelne unabhängig voneinander lauffähige sogenannte Microservices unterteilt, die wiederum auf mehrere Server verteilt und repliziert werden [LeFo14]. Die in dieser Arbeit vorgestellte Methode und Architektur des Prototyps kann für beide Architekturstile verwendet werden. Zur Durchführung der Evaluation ist entscheidend, dass IT-Service-Komponenten unabhängig voneinander lauffähig sind, auf verschiedenen Servern betrieben werden, und damit auch unabhängig voneinander bestehende Log-Dateien ausgewertet werden können. Dies ist mit der verwendeten Versuchsumgebung gegeben. Nachdem die Versuchsumgebung beschrieben wurde, werden nachfolgend die durchgeführten Experimente beschrieben.

8.2 Experimente

Zur Evaluation der Methode werden insgesamt drei verschiedene Experimente durchgeführt. In den ersten beiden Experimenten werden beispielhafte Wartungen durchgeführt, um die Fehlererkennung zu evaluieren. Dazu werden während der Wartungsdurchführung typische Fehler vorgenommen wie sie in Kapitel 3.3 beschrieben sind. Bei der Erstellung der Wartungspläne für die Experimente wurde darauf geachtet, dass alle Pattern mindestens einmal verwendet werden und ein Wartungsszenario beschrieben wird, wie es auch in der Praxis vorkommen kann, bzw. schon in anderen Studien eingesetzt wurde (vgl. [NOBM04]). Der Fokus der beiden Experimente ist dabei jeweils ein anderer. Im ersten Experiment liegt der Fokus auf nebenläufigen Aktivitäten. Im zweiten Experiment liegt der Fokus auf alternativen Aktivitäten. Außerdem werden im dritten Experiment zusätzlich doppelte Aktivitäten verwendet, um Schwachstellen der Methode zu identifizieren.

Im dritten Experiment wird evaluiert, ob mit der vorgestellten Methode und Architektur die Anforderung zur Erkennung von Fehlern online während der Wartungsdurchführung erfüllt werden kann (siehe Kapitel 7.1). Nachfolgend werden die drei Experimente im Detail beschrieben.

8.2.1 Experiment 1: Hinzufügen eines neuen Datenbankservers

In diesem Experiment soll der aktive Datenbankserver der in Kapitel 8.1 vorgestellten Versuchsumgebung durch den zweiten Datenbankserver ausgetauscht werden. Dazu wird zunächst der zweite Datenbankserver gestartet. Anschließend wird der Loadbalancer gestoppt, damit keine Anfragen mehr an die beiden SugarCRM-Instanzen weitergeleitet werden. Sobald der Loadbalancer gestoppt ist, kann die Konfiguration der beiden SugarCRM-Instanzen angepasst werden, so dass keine Verbindung mehr zum ersten Datenbankserver, sondern nur noch zum zweiten Datenbankserver aufgebaut wird. Damit die Konfigurationsänderungen aktiv werden, ist ein Neustart der Webserver notwendig, auf denen die SugarCRM-Instanzen gehostet werden. Das hierfür notwendige vorherige Stoppen der Webserver kann nebenläufig zur Konfigurationsänderung vorgenommen werden, da die Konfigurationsanpassung direkt im jeweiligen Konfigurationsdokument von SugarCRM vorgenommen wird. Sobald eine SugarCRM-Instanz konfiguriert und der jeweilige Webserver gestoppt ist, kann der Webserver gestartet werden, um die neue Konfiguration zu aktivieren. Der erste Datenbankserver kann gestoppt werden, sobald beide Webserver gestoppt sind. Wenn beide Webserver neu gestartet sind und der erste Datenbankserver gestoppt ist, kann der Loadbalancer gestartet werden, um Anfragen an die beiden SugarCRM-Instanzen weiterzuleiten. Abbildung 8.2 zeigt den Wartungsplan für Experiment 1. In diesem Experiment kommen die Pattern NACHFOLGER, DIREKTER NACHFOLGER, VORGÄNGER, DIREKTER VORGÄNGER, ANWESENHEIT, ABWESENHEIT, WERT und ZUSTAND zum Einsatz. Das Experiment zeichnet sich insbesondere durch viele Aktivitäten aus, die nebenläufig zueinander[8] durchgeführt werden können.

[8] Mit „nebenläufig zueinander“ wird in diesem Experiment verstanden, dass diese zeitlich unabhängig voneinander und in unterschiedlicher Reihenfolge durchgeführt werden. Es ist nicht gemeint, dass diese zeigleich parallel zueinander durchgeführt werden.

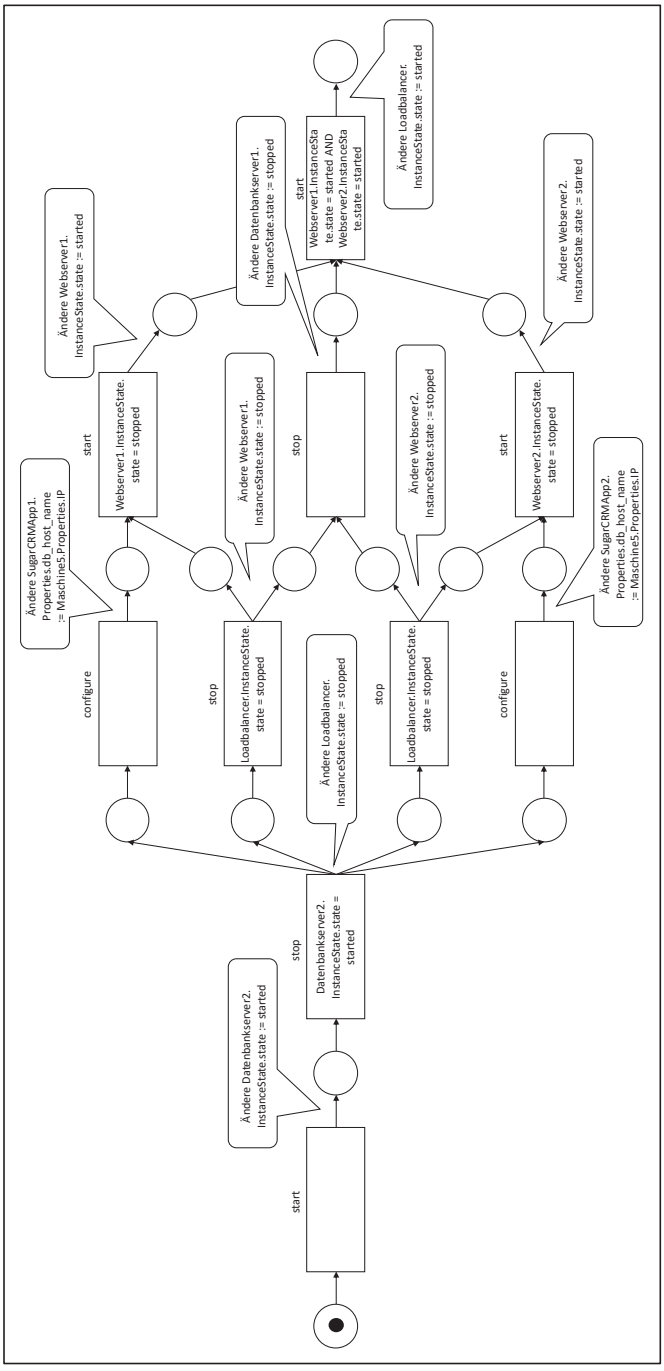


Abbildung 8.2: Wartungsplan für Experiment 1

Zur Evaluation der Methode wird dieses Experiment mehrfach durchgeführt, um die Erkennung verschiedener Fehler und Kombinationen von Fehlern zu testen. Die durchgeführten Fehler können Tabelle 8.1 entnommen werden. Im Rahmen des Experiments werden zunächst Versuchsdurchläufe durchgeführt, in denen nur ein Fehler vorgenommen wird. Es wird jeder Typ von Wartungsfehlern, die in Kapitel 3.3 vorgetellt wurden, in mindestens einem Versuchsdurchlauf ausgeführt. Anschließend werden Versuchsdurchläufe vorgenommen, in denen zwei bzw. drei verschiedene Fehlertypen kombiniert durchgeführt werden. Dadurch können Interdependenzen beim Auftreten mehrerer Fehler untersucht werden, um mögliche Schwachstellen der Methode zu identifizieren.

Tabelle 8.1: Durchgeführte Fehler in Experiment 1

Nr.	Fehlertyp	Durchgeführter Fehler	Fehlerbehandlung
1	Aktivität vergessen	Es wird vergessen, SugarCRMApp1 zu konfigurieren.	SugarCRMApp1 wird konfiguriert. Damit die Konfiguration aktiv wird, wird Webserver1 gestoppt und wieder gestartet.
2	Aktivität in der falschen Reihenfolge durchgeführt	Der Loadbalancer wird gestartet, bevor Webserver2 gestartet wird.	Keine Fehlerbehandlung notwendig, da nach dem Start von Webserver2 der Fehlerzustand aufgehoben ist.
3	Unnötige Aktivität durchgeführt	Bei der Konfiguration von SugarCRMApp1 wird nicht nur die IP geändert, sondern es wird auch ein neuer User angegeben, obwohl dieser mit dem neuen Datenbankserver gleich bleibt.	Für den Parameter User wird der Wert vor der Änderung eingetragen.
4	Aktivität falsch durchgeführt	Anstatt Datenbankserver1 zu stoppen wird Datenbankserver2 gestoppt.	Datenbankserver2 wird gestartet und anschließend wird Datenbankserver1 gestoppt.
5	Umgebungs-inkonsistenz	Es wird die falsche IP-Adresse bei der Konfiguation von SugarCRMApp1 angegeben.	Es wird die IP-Adresse eingetragen, die gemäß Wartungsplan anzugeben ist.

6	Typo	Bei der Angabe der neuen IP bei SugarCRMApp2 wird statt dem Format xxx.xxx.xxx.xxx für die IP das Format xxx.xxx.xxxxxx eingegeben.	Der fehlende Punkt wird nachträglich eingetragen.
7	Fehlerhafte Ausführung einer Aktivität aufgrund eines Software-Bugs oder eines Hardware-Defekts	Webserver2 erhält den Befehl zum Starten, jedoch wird nach dem Start die dazugehörige EC2-Maschine gestoppt, so dass Webserver2 nicht mehr erreichbar ist. Hiermit wird ein Fehler simuliert, dass die Software einen Bug enthält und deswegen der Start nicht ordnungsgemäß durchgeführt werden kann oder dass während der Durchführung der IT-Service-Wartung ein Hardware-Defekt auftritt, der die erfolgreiche IT-Service-Wartung verhindert.	Die EC2-Maschine wird wieder gestartet und anschließend Webserver2 gestartet.
8	Aktivität vergessen & Aktivität in der falschen Reihenfolge durchgeführt	Es wird vergessen, SugarCRMApp1 zu konfigurieren, und der Loadbalancer wird gestartet, bevor Webserver2 gestartet wird.	SugarCRMApp1 wird konfiguriert. Damit die Konfiguration aktiv wird, wird Webserver1 gestoppt und wieder gestartet. Beim zweiten Fehler ist keine Fehlerbehandlung notwendig, da der Fehlerzustand durch den Start von Webserver2 aufgehoben wird.

9	Unnötige Aktivität durchgeführt & Aktivität falsch durchgeführt	Bei der Konfiguration von SugarCRMApp1 wird nicht nur die IP geändert, sondern es wird auch ein neuer User angegeben, obwohl dieser mit der neuen Datenbank gleich bleibt. Zusätzlich wird Datenbankserver2 anstatt Datenbankserver1 gestoppt.	Für den Parameter User wird der Wert vor der Änderung eingetragen. Datenbankserver2 wird gestartet und anschließend wird Datenbankserver1 gestoppt.
10	Unnötige Aktivität durchgeführt & Aktivität falsch durchgeführt & Typo	Bei der Konfiguration von SugarCRMApp1 wird nicht nur die IP geändert, sondern es wird auch ein neuer User angegeben, obwohl dieser mit der neuen Datenbank gleich bleibt. Zusätzlich wird Datenbankserver2 anstatt Datenbankserver1 gestoppt. Außerdem wird die IP-Adresse bei SugarCRMApp2 falsch eingegeben.	Für den Parameter User wird der Wert vor der Änderung eingetragen. Datenbankserver2 wird gestartet und anschließend wird Datenbankserver1 gestoppt. Nachdem der Fehler gemeldet wurde, dass die IP-Adresse bei SugarCRMApp2 falsch eingegeben wurde, wird diese korrigiert.

Die ersten zehn Versuchsdurchläufe werden so durchgeführt, dass bei einer Fehlererkennung der Fehler nicht direkt behoben wird, sondern die Wartungen wie geplant zu Ende durchgeführt werden. Dieses Szenario entspricht einem Szenario, wie es in Betrieben vergefunden wird, wenn Fehler unabsichtlich durchgeführt werden und nicht erkannt werden. Anschließend werden die neun Versuchsdurchläufe wiederholt, jedoch werden die Fehler nach Entdeckung des Fehlers sofort behoben. Es sei darauf hingewiesen, dass die Behebung des Fehlers im betrieblichen Einsatz ein selbstverständlicher Vorgäng wäre, wenn der Fehler direkt erkannt und behoben werden kann. Im Wartungsplan ist ein solches Verhalten jedoch nicht explizit modelliert, um zu prüfen wie sich die Methode bei nicht modellierten Fehlerbehandlungen verhält. In der betrieblichen Praxis kann ein solches Szenario ebenfalls eintreten, wenn beispielsweise zugunsten der Einfachheit und Lesbarkeit eines Wartungsplans auf eine explizite Fehlermodellierung verzichtet wird, oder diese schlicht vergessen wird. Im zweiten Experiment wird ein Wartungsplan mit einer expliziten Fehlerbehandlung verwendet.

8.2.2 Experiment 2: Update einer Anwendung ohne Downtime

In diesem Experiment sollen die beiden SugarCRM-Instanzen nacheinander neu konfiguriert werden, jedoch ohne dafür eine gleichzeitige Downtime der beiden Instanzen zu benötigen. Dafür wird zunächst der Loadbalancer so konfiguriert, dass Anfragen nur noch an eine der beiden SugarCRM-Instanzen weitergeleitet werden. Um die neue Konfiguration aktiv zu schalten, wird die Konfiguration des Loadbalancers anschließend erneut geladen werden. Sobald die neue Konfiguration des Loadbalancers geladen ist, kann der Webserver, der keine Anfragen mehr vom Loadbalancer erhält, gestoppt werden. Nebenläufig dazu kann die Konfiguration der jeweiligen SugarCRM-Instanz angepasst werden. Sobald die Konfiguration durchgeführt wurde und der Webserver gestoppt ist, kann der Webserver wieder neu gestartet werden, um die neue Konfiguration zu aktivieren.

Sollte der Webserver nicht neu gestartet werden können, muss eine Fehlerbehandlung durchgeführt werden. Hierfür musst zunächst einer der beiden Webserver, Webserver3 bzw. Webserver4, gestartet werden. Anschließend wird der Loadbalancer so konfiguriert werden, dass Anfragen an den zuvor gestarteten Webserver geleitet werden und die Konfiguration des Loadbalancers durch den Befehl „reload“ neu geladen wird. Wenn die Fehlerbehandlung durchgeführt ist, wird die Wartung beendet. In einem realen Szenario würde anschließend zunächst die Fehlerursache analysiert werden, bevor die Wartung wiederholt wird. Grund für die Fehlerursache könnte in diesem Fall z.B. ein Bug in der Software sein. Durch Anwendung der in dieser Arbeit vorgestellten Methode würde bereits ausgeschlossen werden, dass ein Durchführungsfehler die Ursache für den fehlerhaften Start des Webservers ist.

Wenn der Webserver erfolgreich neu gestartet wurde, dann ist keine Fehlerbehandlung notwendig. Stattdessen wird die Konfiguration des Loadbalancers aktualisiert, sodass die Anfragen an die bereits neu konfigurierte SugarCRM-Instanz geleitet werden und keine Anfragen mehr an die zweite SugarCRM-Instanz geleitet werden. Anschließend wird die zweite SugarCRM-Instanz nach demselben Vorgehen wie die erste SugarCRM-Instanz gewartet. Abbildung 8.3 zeigt den Wartungsplan für Experiment 2. In diesem Experiment werden im Vergleich zu Experiment 1 zusätzlich die Pattern ALTERNATIVE ANWESENHEIT und ALTERNATIVE ABWESENHEIT verwendet, da XOR-Schaltungen im Wartungsplan verwendet werden. Ein weiterer Unterschied zu Experiment 1 ist die Verwendung von mehrfachen Aktivitäten (reload des Loadbalancers). Im Rahmen des Experiments wird somit geprüft, inwiefern die Methode mit diesen mehrfachen Aktivitäten umgehen kann oder ob Falschmeldungen erzeugt werden.

Dieses Experiment wurde insgesamt 20 Mal durchgeführt. Wie bei Experiment 1 wurde in den ersten zehn Versuchsdurchläufen ein Fehler durchgeführt, ohne ihn anschließend zu korrigieren. Daraufhin wurden die Versuchsdurchläufe wiederholt, jedoch wurden die Fehler nach ihrer Erkennung korrigiert. Für eine Übersicht der in Experiment 2 durchgeführten Fehler siehe Tabelle 8.2.

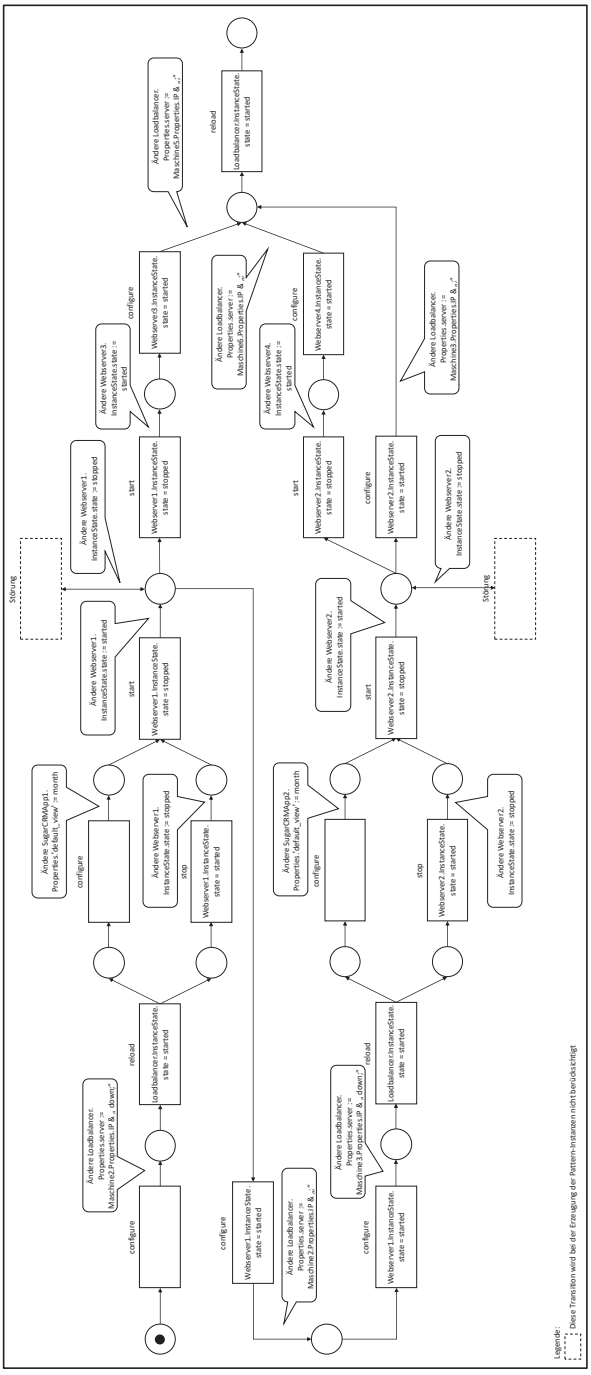


Abbildung 8.3: Wartungsplan für Experiment 2

Tabelle 8.2: Durchgeführte Fehler in Experiment 2

Nr.	Fehlertyp	Durchgeführter Fehler	Fehlerbehandlung
1	Aktivität vergessen	Es wird vergessen, den Loadbalancer zu konfigurieren und die Konfiguration neu zu laden, bevor Webserver1 konfiguriert oder gestoppt wird.	Nachdem der Fehler entdeckt wurde, wird der Loadbalancer nachträglich konfiguriert. Anschließend wird die Wartung fortgesetzt.
2	Aktivät in der falschen Reihenfolge durchgeführt	Webserver1 und Webserver2 werden beide zur selben Zeit gestoppt, anstatt nur einen zu stoppen.	Nachdem der Fehler gemeldet wurde, wird Webserver2 umgehend wieder gestartet.
3	Unnötige Aktivität durchgeführt	Obwohl Webserver1 korrekt gestartet wurde und der Loadbalancer wieder so konfiguriert ist, dass Anfragen an SugarCRMApp1 geleitet werden, wird Webserver3 gestartet.	Webserver3 wird nach der Fehlererkennung wieder gestoppt.
4	Aktivität falsch durchgeführt	Im Loadbalancer wird konfiguriert, dass keine Anfragen mehr an Webserver1 geleitet werden sollen. Anstatt anschließend Webserver1 zu stoppen, wird Webserver2 gestoppt und konfiguriert. Dasselbe passiert anschließend anders herum, so dass eigentlich Webserver2 gestoppt werden sollte. Stattdessen wird jedoch Webserver 1 gestoppt.	Webserver2 wird wieder gestartet. Anschließend wird die Wartung mit dem Stoppen und Konfigurieren von Webserver1 fortgeführt.
5	Lokale Inkonsistenz	Nach dem Start von Webserver1 wird der Loadbalancer falsch konfiguriert, so dass Anfragen nicht an Webserver1 sondern	Der Loadbalancer wird umgehend korrekt konfiguriert, so dass Anfragen wieder an Webserver1 anstatt an

		stattdessen an Webserver4 gesendet werden.	Webserver4 gesendet werden.
6	Typo	Bei der Konfiguration von Webserver1 wird der Wert „monht“ anstatt „month“ angegeben.	Nach der Fehlererkennung wird Webserver1 erneut konfiguriert.
7	Fehlerhafte Ausführung einer Aktivität aufgrund eines Software-Bugs oder eines Hardware-Defekts	Im Loadbalancer wird konfiguriert, dass Anfragen nur noch an Webserver1 geleitet werden sollen. Währenddessen wird jedoch die EC2-Maschine des Webservers gestoppt, so dass Webserver1 nicht mehr erreichbar ist.	Es wird gemerkt, dass der Webserver1 nicht mehr erreichbar ist und daher vor der Konfiguration des Loadbalancers direkt Webserver3 gestartet.
8	Aktivität vergessen & Aktivität in der falschen Reihenfolge durchgeführt	Es wird vergessen den Loadbalancer zu konfigurieren und die Konfiguration neu zu laden, bevor Webserver1 konfiguriert wird. Zusätzlich werden Webserver1 und Webserver2 zur selben Zeit gestoppt, anstatt nur einen Webserver zu stoppen.	Die Konfiguration des Loadbalancers wird nachgeholt. Webserver2 wird wieder gestartet, nachdem der Fehler entdeckt wurde.
9	Aktivität falsch durchgeführt & unnötige Aktivität durchgeführt	Im Loadbalancer wird konfiguriert, dass keine Anfragen mehr an Webserver1 geleitet werden sollen. Anstatt anschließend Webserver1 zu stoppen, wird jedoch Webserver2 gestoppt und konfiguriert. Dasselbe passiert anschließend anders herum, so dass eigentlich Webserver2 gestoppt werden sollte. Stattdessen wird jedoch Webserver1 gestoppt. Als unnötige Aktivität wird	Webserver2 wird wieder gestartet und Webserver1 wird gestoppt. Wenn anschließend im weiteren Verlauf der Wartung Webserver1 anstatt Webserver2 gestoppt wird, wird Webeserver1 wieder gestartet und Webserver2 gestoppt. Wenn Webserver3 gestartet wird, wird Webserver3 nach der Erkennung des Fehlers wieder gestoppt.

		außerdem Webserver3 nach der Konfiguration des Loadbalancers, wenn Anfragen wieder an Webserver1 gesendet werden, gestartet.	
10	Unnötige Aktivität durchgeführt & Typo & Aktivität falsch durchgeführt	Obwohl Webserver1 korrekt gestartet wurde und der Loadbalancer wieder so konfiguriert ist, dass Anfragen an SugarCRMApp1 geleitet werden, wird Webserver3 gestartet. Bei der Konfiguration von Webserver2 wird anschließend der Wert „monht“ anstatt „month“ angegeben. Anschließend wird der Loadbalancer so konfiguriert, dass Anfragen an Webserver4 anstatt an Webserver2 weitergeleitet werden.	Webserver3 wird gestoppt. Wenn der falsch eingegebene Wert „month“ erkannt wird, wird der Wert korrigiert. Nach der Erkennung der falschen Konfiguration des Loadbalancers wird diese korrigiert, so dass der Loadbalancer Anfragen an Webserver2 anstatt an Webserver4 sendet.

8.2.3 Experiment 3: Performance

Im dritten Experiment wird evaluiert, ob mit der in Kapitel 7.2 vorgeschlagenen Architektur die Anforderung nach einer Fehlererkennung online während der Wartungsdurchführung erfüllt werden kann. Zur Überprüfung dieser Anforderung werden Vorgang-Ereignisse erzeugt, an die Prüfeinheit gesendet und anschließend die Antwortzeit der Prüfeinheit für die Auswertung der Anti-Pattern gemessen. Die Vorgang-Ereignisse werden mithilfe der Open Source Software SoapUI[9] erzeugt, die es erlaubt, die Anzahl der erzeugten Vorgang-Ereignisse und den zeitlichen Abstand des Versands der einzelnen Vorgang-Ereignisse nach Bedarf zu steuern. SoapUI und die Prüfeinheit wurden beide auf derselben Maschine[10] ausgeführt, wodurch bei der Performance-Messung Latenzen des Netzwerks sowie die Performance des Log-Filters und der Log-Agenten nicht berücksichtigt werden. Die Performance des Log-Filters und der Log-Agenten wird nicht gemessen, da Log-Agenten

[9] https://www.soapui.org
[10] Betriebssystem: macOS 10.13.4, CPU: 2,3 GHz Intel Core i7, RAM: 8 GB 1600 MHz DDR3

immer nur die Vorgang-Ereignisse einzelner Komponenten erfassen und versenden müssen. Dadurch ist die Menge der zu verarbeitenden Ereignisse im Vergleich zur Prüfeinheit gering und nicht der Performance-limitierende Faktor in der Architektur. Der Log-Filter muss im Gegensatz zu den Log-Agenten Vorgang-Ereignisse mehrerer Komponenten verarbeiten. Da Log-Filter jedoch redundant ausgelegt werden können und dadurch die Menge der zu verarbeitenden Vorgang-Ereignisse skalierbar ist, stellt der Log-Filter ebenfalls keinen Performance-limitierenden Faktor in der Architektur dar.

Für die Messung der Performance der Anti-Pattern-Auswertung sind zwei Faktoren entscheidend: zum einen die Anzahl eintreffender Vorgang-Ereignisse, zum anderen die Komplexität des Wartungsplans, d.h. wie viele Aktivitäten bzw. Pattern-Instanzen der Wartungsplan enthält. Zur Messung dieser beiden Faktoren wurde zunächst ein Testlauf mit dem Wartungsplan aus Experiment 2 durchgeführt. Dieser Wartungsplan ist aus Sicht der Anzahl der Aktivitäten (17 Aktivitäten) im Vergleich zum Wartungsplan aus Experiment 1 komplexer und enthält alle möglichen Pattern-Instanzen. In SoapUI wurden alle 13 Vorgang-Ereignisse für den Pfad zur Konfiguration von Webserver1 und Webserver2 hinterlegt. Im ersten Testlauf wurden insgesamt 10 Threads in SoapUI erstellt, d.h. es wurden 10 virtuelle User erzeugt, die gleichzeitig jeweils die 13 Vorgang-Ereignisse an die Prüfeinheit senden. Somit wurden in der Prüfeinheit insgesamt 130 Vorgang-Ereignisse gleichzeitig gegen den Wartungsplan ausgewertet. In einem zweiten und dritten Testlauf wurde die Anzahl der Threads in SoapUI auf 100 bzw. 1000 erhöht, wodurch insgesamt 1.300 bzw. 13.000 Vorgang-Ereignisse gleichzeitig an die Prüfeinheit gesendet wurden.

Diese drei Testläufe wurden anschließend wiederholt, jedoch wurde der Wartungsplan 10 Mal bzw. 100 Mal in der Prüfeinheit geladen, anstatt nur einmal. Dadurch hat sich die Komplexität des auszuwertenden Wartungplans auf 170 bzw. 1.700 Aktivitäten erhöht. Tabelle 8.3 gibt einen Überblick über die durchgeführten Testläufe für die Performance-Messung.

Tabelle 8.3: Übersicht Experiment 3

Testlauf	Anzahl Wartungspläne/ Anzahl Aktivitäten	Anzahl Threads/ Anzahl Vorgang-Ereignisse
1	1/ 17	10/ 130
2	1/ 17	100/ 1.300
3	1/ 17	1.000/ 13.000
4	10/ 170	10/ 130
5	10/ 170	100/ 1.300
6	10/ 170	1.000/ 13.000
7	100/ 1.700	10/ 130
8	100/ 1.700	100/ 1.300
9	100/ 1.700	1.000/ 13.000

8.3 Ergebnisse

Nachdem die Experimente im Detail beschrieben wurden, werden nachfolgend die Ergebnisse der drei Experimente dokumentiert. Für die Experimente eins und zwei werden zunächst die Anzahl der vom Prototyp gemeldeten Fehler je Versuchsdurchlauf in einer Konfusionsmatrix aufgelistet, wie sie z.B. zur Evaluation bei Machine Learning Experimenten üblich ist [Powe11]. Die Konfusionsmatrix ist wie in Tabelle 8.4 dargestellt aufgebaut und wie folgt zu lesen.

Tabelle 8.4: Konfusionsmatrix

	Fehler gemeldet	Fehler nicht gemeldet
Injizierter Fehler	TP	TN
Kein Fehler	FP	FN

TP (true positive) gibt die Anzahl der Fehlermeldungen wieder, die tatsächlich einen Fehler gemeldet haben. FP (false positive) gibt die Fehler wieder, die einen Fehler gemeldet haben, obwohl kein Fehler auftreten ist. TN (true negative) gibt die Anzahl nicht gemeldeter Fehler wieder, obwohl ein Fehler aufgetreten ist. FN (false negative) gibt die Anzahl der nicht gemeldeten Fehler wieder, bei denen auch kein Fehler aufgetreten ist. Diese Kennzahl ist zur Evaluation dieser Methode nicht weiter relevant und wird deswegen nur der Vollständigkeit wegen erwähnt. Auf Basis dieser Kennzahlen werden anschließend die

Evaluationsmetriken Recall, Precision und das harmonische Mittel aus Recall und Precision, der F-Score berechnet[11].

Die Metrik Recall gibt dabei das Verhältnis der gemeldeten Fehler zu den tatsächlich stattgefunden Fehler an und wird wie folgt berechnet:

$$Recall = \frac{TP}{TP + TN}$$

Die Metrik Precision gibt das Verhältnis aller relevanten gemeldeten Fehlermeldungen zu allen gemeldeten Fehlermeldungen an und wird wie folgt berechnet:

$$Precision = \frac{TP}{TP + FP}$$

Eine isolierte Betrachtung der beiden Metriken Recall und Precision ist für die Beurteilung der Methode zur Fehlererkennung nicht ausreichend, da beispielsweise ein Recall-Wert von eins alleine dadurch erreicht werden kann, dass jede durchgeführte und vergessene Aktivität in einer IT-Service-Wartung als Fehler gemeldet wird. In diesem Fall würden zwar auch die tatsächlichen Fehler erkannt werden, jedoch würden so viele Fehler gemeldet werden, dass die tatsächlich stattgefunden Fehler unter dieser Informationsflut leicht übersehen werden können. Die Metrik Precision hingegen kann dadurch verbessert werden, dass nur sehr wenige Fehler gemeldet werden, die jedoch dafür nur tatsächlich gefundene Fehler melden. Um diese isolierte Betrachtungsweise zu vermeiden, kann der F-Score verwendet werden. Der F-Score setzt die beiden Metriken Precision und Recall ins Verhältnis und wird wie folgt berechnet:

$$F - Score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

Wenn eine der beiden Metriken stärker gewichtet werden soll, kann zusätzlich eine Gwichtung eingeführt werden. Der gewichtete F-Score kann wie folgt berechnet werden:

$$F - Score_{gewichtet} = \frac{2}{\alpha \frac{1}{Precision} + (1 - \alpha) \frac{1}{Recall}}$$

Im Fall der Fehlererkennung kann argumentiert werden, dass der Recall höher gewichtet werden soll als die Precision, da bereits ein übersehener Fehler gravierende wirtschaftliche

[11] Für eine weiterführende Beschreibung der Evaluationsmetriken Recall, Precision, F-Score und weiterer Metriken siehe [Powe11].

Folgen haben kann, wie die Beispiele in Kapitel 1 zeigen. Allerdings können zuviele Fehlermeldungen auch dazu führen, dass die Anwender, die die Durchführung einer IT-Service-Wartung überwachen, Fehlermeldungen nicht mehr ernst nehmen, womit das Ziel der Fehlererkennung verfehlt werden würde. Aus diesem Grund werden die Metriken Precision und Recall in dieser Arbeit gleich gewichtet.

In allen vorgestellten drei Metriken werden doppelte Fehlermeldungen nicht betrachtet. Doppelte Fehlermeldungen können beim Einsatz von Anti-Pattern jedoch leicht auftauchen. Zur Veranschaulichung soll folgendes Beispiel dienen: Eine IT-Service-Wartung besteht aus einer Sequenz (a, b, c). Zur Überwachung dieser Sequenz würden die drei Anti-Pattern DIREKTER NACHFOLGER, DIREKTER VORGÄNGER und ANWESENHEIT verwendet werden. Wenn bei der Ausführung der IT-Service-Wartung nun b vergessen wird und nur die Sequenz (a, c) ausgeführt wird, würden alle drei Anti-Pattern einen Fehler melden. Das Anti-Pattern DIREKTER NACHFOLGER würde den Fehler melden, dass b nicht nach a ausgeführt wurde. Das Anti-Pattern DIREKTER VORGÄNGER würde den Fehler melden, dass b nicht vor a ausgeführt wurde und das Anti-Pattern ANWESENHEIT würde den Fehler melden, dass b nicht durchgeführt wurde. Für die drei Metriken oben würden die drei Fehlermeldungen als eine Fehlermeldung gezählt werden, da ansonsten die Metrik Precision positiv verfälscht werden würde, ohne dass sich die Metrik tatsächlich durch die vielen Fehlermeldungen verbessert hätte.

Um bei der Evaluation eine Aussage über die Anzahl an Fehlermeldungen je tatsächlich durchgeführtem Fehler treffen zu können, wird die Metrik „Informationsgewinn (IG)“ eingeführt. Die Metrik IG gibt an, wie wertvoll eine Fehlermeldung ist. Bei einer zunehmenden Anzahl an Fehlermeldungen zum selben Fehlergrund, wie im Beispiel oben, nimmt der Nutzen einer Fehlermeldung ab, da die Fehlerursache bereits bekannt ist und jede weitere Fehlermeldung zum selben Fehlergrund daher unnötig ist.[12] Die Metrik wird wie folgt berechnet:

$$IG = \frac{TP}{Gesamtanzahl\ Fehlermeldungen\ für\ TP}$$

Der Vollständigkeit wegen wird bei jedem Experiment zusätzlich die Anzahl aller Fehlermeldungen angegeben, unabhängig davon, ob es sich um eine TP- oder FP-Fehlermeldung handelt.

[12] Für den betrieblichen Einsatz können solche mehrfachen Fehlermeldungen reduziert werden, indem, wie in Kapitel 8.4 näher beschrieben wird, ein zusätzlicher Filter in der Complex Event Processing Engine eingebaut wird.

8.3.1 Ergebnisse Experiment 1

Für Experiment 1 werden in Tabelle 8.5 die Anzahl aller gemeldeten Fehlermeldungen dargestellt. Nachfolgend werden die Ergebnisse im Detail beschrieben.

Tabelle 8.5: Übersicht der Fehlermeldungen für Experiment 1

Nr.	Fehler gemeldet		Fehler nicht gemeldet	Insgesamt gemeldete TP-Fehler	Insgesamt gemeldete Fehler
	TP	FP	TN		
1	1	0	0	4	4
2	1	0	0	4	4
3	1	0	0	1	1
4	2	0	0	5	5
5	1	0	0	5	5
6	1	0	0	5	5
7	1	0	0	1	1
8	2	0	0	7	7
9	3	0	0	6	6
10	4	0	0	7	7
11-1	1	2	0	2	4
11-2	1	2 + 1	0	2	5
12	1	0	0	4	4
13	1	1	0	1	2
14	1	1 + 1	0	1	3
15	1	1	0	1	2
16	1	1	0	1	2
17	1	1	0	1	2
18-1	2	2	0	6	8
18-2	2	2 + 1	0	6	9
19	2	2 + 1	0	2	5
20	3	2 + 1	0	3	6

In allen Versuchsläufen wurden alle injizierten Fehler erkannt. In den ersten zehn Versuchsläufen, in denen die durchgeführten Fehler nicht direkt behoben wurden, wurden zudem keine Fehlermeldungen fälschlicherweise gemeldet. Bei den Versuchsläufen 1, 2, 4,

8, 9 und 10 wurden mehr Fehler gemeldet, als es Fehlerursachen gab. Beispielhaft für diese Versuchsläufe wird Versuchslauf 1 genauer beschrieben.

In Versuchslauf 1 wird als Fehler vergessen, SugarCRMApp1 zu konfigurieren. Wenn bei der IT-Service-Wartung nun Webserver1 gestartet wird, dann melden die Anti-Pattern NACHFOLGER und VORGÄNGER den Fehler, dass SugarCRMApp1 nicht konfiguriert wurde. NACHFOLGER meldet einen Fehler, da nach dem Stoppen des Loadbalancers und vor dem Starten von Webserver1 SugarCRMApp1 konfiguriert sein muss (Fehlermeldung 1). VORGÄNGER meldet einen Fehler, da beim Starten von Webserver1 noch nicht SugarCRMApp1 konfiguriert wurde (Fehlermeldung 2). Wenn die IT-Service-Wartung fortgeführt wird, dann wird eine erneute Fehlermeldung erzeugt, wenn der Loadbalancer wieder gestartet wird. Auslöser für diese Fehlermeldung ist wiederum das Anti-Pattern NACHFOLGER, da SugarCRMApp1 nach dem Stoppen des Loadbalancers und vor dem Starten des Loadbalancers durchgeführt sein muss (Fehlermeldung 3). Wenn das Anti-Pattern NACHFOLGER das Starten des Loadbalancers erkennt, wird jedoch kein Ereignis in der Vorgang-Ereignis-Historie gefunden, die der Aktivität zum Konfigurieren von SugarCRMApp1 entspricht. Die vierte Fehlermeldung wird aufgrund des Anti-Patterns ANWESENHEIT erzeugt. Nach Ablauf des Wartungsfensters wird in der In Memory Datenbank keine Aktivität im Zeitraum des Wartungsfensters gefunden, die der Konfiguration von SugarCRMApp1 entspricht, weshalb erneut eine Fehlermeldung erzeugt wird (Fehlermeldung 4).

Bei den Versuchsläufen 11 bis 20, bei denen die Fehler nach Erkennung umgehend behoben wurden, fallen die doppelten Fehlermeldungen deutlich geringer aus. Der Grund hierfür ist, dass die Anti-Pattern NACHFOLGER und ANWESENHEIT nach direkter Behebung des Fehlers nicht mehr für den ursprünglichen Fehlergrund auftreten. Die Anzahl der fälschlicherweise gemeldeten Fehler (FP) ist dafür jedoch angestiegen. Der Grund hierfür ist, dass bei der Behebung eines Fehlers die IT-Service-Wartung vom Wartungsplan abweicht, wenn die Fehlerbehebung nicht explizit im Wartungsplan modelliert ist. In Tabelle 19 wird die Anzahl der falsch gemeldeten Fehler in der Spalte FB aufgelistet. Dabei werden teilweise Summen angezeigt, wie beispielsweise bei Versuchsduchlauf 14. Der Grund hierfür ist, dass in der Tabelle unterschieden wird zwischen „echten“ Fehlermeldungen und Fehlermeldungen, die aufgrund einer fehlenden modellierten Fehlerbehandlung erzeugt werden. Fehlermeldungen, die aufgrund einer fehlenden modellierten Fehlerbehandlung im Wartungsplan erzeugt werden, stehen im Term an erster Position. „Echte“ Fehlermeldungen stehen an zweiter Position. Zur Veranschauchlichung wird Versuchs-durchlauf 14 beschrieben. Bei Versuchsdurchlauf 14 wird anstatt Datenbankserver1 Datenbankserver2 gestoppt. Beim Stoppen von Datenbankserver2 wird das Anti-Pattern

ABWESENHEIT aktiviert, da im Wartungsplan ein Stoppen von Datenbankserver2 nicht vorgesehen ist (Fehlermeldung 1). Wenn im Anschluss Datenbankserver2 wieder gestartet wird, erscheint eine Fehlermeldung, dass Datenbankserver2 zu oft gestartet wurde. Diese Fehlermeldung wird wiederum durch das Anti-Pattern ABWESENHEIT erzeugt, da ein Start von Datenbankserver2 im Wartungsplan nur ein einziges Mal vorgesehen ist (Fehlermeldung 2). Diese Meldung erscheint aufgrund der nicht modellierten Fehlerbehandlung. Im weiteren Verlauf der IT-Service-Wartung wird anschließend tatsächlich Datenbankserver1 gestoppt. Nun erzeugt das Anti-Pattern DIREKTER NACHFOLGER eine „echte“ Fehlermeldung, dass der Loadbalancer nach dem Start von Datenbankserver2 nicht gestoppt wurde (Fehlermeldung 3). Der Grund hierfür ist, dass gemäß Wartungsplan nach dem Start von Datenbankserver2 direkt im Anschluss der Loadbalancer gestoppt werden muss.

In Tabelle 8.5 sind die Versuchsdurchläufe 11 und 18 jeweils doppelt erfasst. Der Grund ist, dass bei der Evaluation eine unterschiedliche Anzahl an Fehlermeldungen aufgetreten ist, je nachdem in welcher Reihenfolge die Aktivitäten im Wartungsplan durchgeführt wurden. Zur Veranschaulichung soll Versuchsdurchlauf 11 dienen. In Versuchdurchlauf 11 wird, analog zu Versuchsdurchlauf 1, vergessen SugarCRMApp1 zu konfigurieren. Beim Starten von Webserver1 melden die Anti-Pattern NACHFOLGER und VORGÄNGER jeweils den Fehler, dass vergessen wurde SugarCRMApp1 zu konfigurieren (Fehlermeldung 1 und 2). Wenn SugarCRMApp1 nachträglich konfiguriert wird und zusätzlich Webserver1 gestartet und gestoppt wird, werden aufgrund des Anti-Patterns ABWESENHEIT zwei Fehlermeldungen erzeugt, dass Webserver1 jeweils zu häufig gestartet bzw. gestoppt wurde (Fehlermeldungen 3 und 4). Diese Fehlermeldungen sind auf die nicht modellierte Fehlerbehandlung zurückzuführen. Eine „echte“ Fehlermeldung kann beim Starten des Loadbalancers durch das Anti-Pattern NACHFOLGER erzeugt werden. Als Fehler wird in diesem Fall gemeldet, dass Datenbankserver1 nicht vor dem Starten des Loadbalancers beendet wurde (Fehlermeldung 5). Diese Fehlermeldung wird erzeugt, wenn Datenbankserver1 nach dem ersten Stoppen von Webserver1 beendet wurde (Versuchsdurchlauf 11-2). Das Anti-Pattern NACHFOLGER wird beim zweiten Stoppen von Webserver1 erneut aktiviert, so dass beim Starten des Loadbalancers in der Vorgang-Ereignis-Historie geprüft wird, ob zwischen dem zweiten Stoppen von Webserver1 und dem Start des Loadbalancers Datenbankserver1 gestoppt wurde (siehe die Beschreibung des Anti-Patterns in Kapitel 5.4.1). Dies ist in diesem Fall jedoch nicht so, da Datenbankserver1 vor dem zweiten Stop von Webserver1 gestoppt wurde. Wird jedoch Datenbankserver1 erst nach dem zweiten Stoppen von Webserver1 gestoppt, wird keine Fehlermeldung erzeugt, da der Stop von Datenbankserver1 zwischen dem Start des Loadbalancers und zwischen den beiden Stops von Webserver1 erfolgt ist (Versuchsdurchlauf 11-1).

Aufgrund der unterschiedlichen Ergebnisse der Versuchsdurchläufe 1-10, sowie 11-20 werden die Kennzahlen Precision, Recall, F-Score und der Informationsgewinn für eine bessere Vergleichbarkeit in Tabelle 8.6 für alle Versuchsdurchläufe und jeweils seperat für die Versuchsdurchläufe 1-10 und 11-20 aufgelistet.

Tabelle 8.6: Metriken für Experiment 1

Versuchsdurchlauf	Precision	Recall	F-Score	IG
1-20	0,607	1	0,756	0,430
1-10	1	1	1	0,347
11-20	0,436	1	0,607	0,567

Der Recall liegt bei allen Versuchsdurchläufen bei 1, d.h. es wurden alle injizierten Fehler erkannt. Bei den Versuchsdurchläufen 1-10 wird der höchstmögliche F-Score von 1 erreicht. Jedoch ist der Informationsgewinn der Fehlermeldungen bei diesen Versuchsdurchläufen dafür am geringsten. Ein Grund hierfür sind die vermehrten Fehlermeldungen aufgrund des Anti-Patterns NACHFOLGER. Bei den Versuchsdurchläufen 11-20 ist der Informationsgewinn der Fehlermeldungen höher, da aufgrund der sofortigen Fehlerbehebung das Anti-Pattern NACHFOLGER nur bei der ersten Aktivität, die nach einer vergessenen Aktivität durchgeführt wird, eine Fehlermeldung erzeugt. Die Precision erreicht bei den Versuchsdurchläufen 1-10 den höchsten Wert. Das ist darauf zurückzuführen, dass bis auf die injizierten Fehler keine Abweichungen bei der Durchführung der IT-Service-Wartung vom Wartungsplan vorgenommen wurden. Bei den Versuchsdurchläufen 11-20 sinkt die Precision auf den Wert 0,436. Der Grund hierfür sind die falschen Fehlermeldungen, die aufgrund der nicht modellierten Fehlerbehandlungen im Wartungsplan erzeugt werden. Wenn der Wartungsplan eine explizite Fehlerbehandlung enthalten würde, beispielsweise dass nach dem Start von Webserver1 erneut SugarCRMApp1 konfiguriert und Webserver1 gestoppt werden dürften, würde die Precision ansteigen, da weniger falsche Fehlermeldungen erzeugt werden würden.

8.3.2 Ergebnisse Experiment 2

Für Experiment 2 werden in Tabelle 8.7 die Anzahl aller gemeldeten Fehlermeldungen dargestellt. Nachfolgend werden die Ergebnisse im Detail beschrieben.

Tabelle 8.7: Übersicht der Fehlermeldungen für Experiment 2

Nr.	Fehler gemeldet		Fehler nicht gemeldet	Insgesamt gemeldete TP-Fehler	Insgesamt gemeldete Fehler
	TP	FP	TN		
1	3	2	0	8	16
2	2	3	0	2	10
3	1	2	0	5	12
4	3	2	1	16	18
5	1	2	0	8	16
6	1	1	0	13	17
7	1	2	0	1	9
8	4	6	0	12	22
9	4	2	1	23	26
10	4	4	0	13	21
11	1	3	0	1	12
12	2	6	0	3	17
13	1	3	0	5	18
14	1	6	0	4	18
15	1	4	0	6	16
16	1	2	0	1	9
17	0	0	0	0	0
18	2	6	0	3	23
19	3	8	0	8	28
20	3	5	0	9	24

In Experiment 2 fällt im Vergleich zu Experiment 1 die deutlich erhöhte Anzahl an falschen Fehlermeldungen, sowie die hohe Zahl an insgesamt gemeldeten Fehlern auf. Zudem gibt es insgesamt zwei Versuchsdurchläufe, bei denen ein Fehler nicht erkannt wurde. Der hauptsächliche Grund für die hohe Anzahl an falschen Fehlermeldungen, sowie der beiden nicht erkannten Fehler sind die doppelten Aktivitäten im Experiment. So treten in diesem Experiment auch falsche Fehlermeldungen auf, wenn gar kein Fehler eingetreten ist. Der Grund für diese falschen Fehlermeldungen ist die doppelte Aktivität „reload des Loadbalancers“ und wird nachfolgend detailliert beschrieben.

Wenn die IT-Service-Wartung gemäß Wartungplan in Abbildung 8.3 durchgeführt wird, wird zunächst der Loadbalancer so konfiguriert, dass keine Anfragen mehr an Webserver1

geleitet werden. Daraufhin wird die Konfiguration des Loadbalancers neu geladen (reload). Anschließend wird Webserver1 konfiguriert und gestoppt werden. Diese Aktivitätenabfolge (reload, konfiguriere Webserver1) bzw. (reload, stoppe Webserver1) wird durch das Anti-Pattern NACHFOLGER geprüft. Das Anti-Pattern wird durch die erste Aktivität der Aktivitätenabfolge, in diesem Fall ist das die Aktivität „reload des Loadbalancers“, aktiviert. Wenn das Anti-Pattern nun eine Aktivität erkennt, die nach dem Stoppen bzw. Konfigurieren von Webserver1 durchgeführt wird, wie beispielsweise das Starten von Webserver1 oder das Stoppen, Konfigurieren und Starten von Webserver2, wird geprüft, ob bereits Webserver1 konfiguriert und gestoppt wurde. Wenn die IT-Service-Wartung nun wie geplant ausgeführt wird, wird beispielsweise nach dem Starten von Webserver1 geprüft, ob Webserver1 konfiguriert und gestoppt wurde. Dies ist bei einer korrekten Durchführung der IT-Service-Wartung der Fall und das Anti-Pattern meldet keinen Fehler. Gemäß des Wartungsplans in Abbildung 8.3 wird nach dem Starten von Webserver1 der Loadbalancer erneut konfiguriert und neu geladen (reload), um anschließend Webserver2 zu konfigurieren und zu stoppen. Beim „reload“ des Loadbalancers wird nun erneut das Anti-Pattern NACHFOLGER für die Aktivitätenabfolge (reload, konfiguriere Webserver1) bzw. (reload, stoppe Webserver1) aktiviert. Da als nächstes jedoch Webserver2 gestoppt und konfiguriert wird und zwischen des „reloads“ des Loadbalancers und dem Stoppen und Konfigurieren von Webserver2, Webserver1 weder gestoppt noch konfiguriert wurde, meldet das Anti-Pattern die Fehler, dass nach dem reload des Loadbalancers vergessen wurde Webserver1 zu konfigurieren bzw. zu stoppen.

Die doppelte Aktivität „reload“ des Loadbalancers ist auch ein Grund für die hohe Anzahl an insgesamt gemeldeten Fehlermeldungen. Da das Anti-Pattern NACHFOLGER jedes Mal eine Fehlermeldung erzeugt, wenn eine nachfolgende Aktivität (konfiguriere Webserver2, stoppe Webserver2, starte Webserver2, starte Webserver4, konfiguriere Loadbalancer) durchgeführt wird. Dadurch werden beispielsweise sechs Fehlermeldungen erzeugt, wenn Webserver2 konfiguriert, gestoppt und gestartet wird.

Eine weitere Auffälligkeit bei der Durchführung von Experiment 2 sind die Versuchsdurchläufe 4 und 9, bei denen ein Fehler nicht erkannt wurde. Der Grund hierfür kann wiederum auf die doppelte Aktivität „reload“ zurückgeführt werden. Bei beiden Versuchsdurchläufen wurde die Reihenfolge des Konfigurierens, Startens und Stoppens der Webserver vertauscht, d.h. zuerst wurde Webserver2 gewartet und anschließend erst Webserver1. Da der Loadbalancer wie geplant konfiguriert wurde, d.h. dass zunächst keine Anfragen an Webserver1 geleitet werden, führt dieser Fehler dazu, dass Anfragen auf Webserver2 geleitet werden. Da jedoch fälschlicherweise Webserver2 anstatt Webserver1 zu diesem Zeitpunkt gewartet wurde, wäre ein Ausfall des gesamten IT-Service eingetreten.

Bei der Durchführung der IT-Service-Wartung wurde beim Konfigurieren von Webserver2 anstatt von Webserver1 eine korrekte Fehlermeldung erzeugt, dass zunächst Webserver1 konfiguriert werden muss. Als Webserver1 anstatt Webserver2 konfiguriert wurde, wurde jedoch keine Fehlermeldung erzeugt, da die Konfiguration von Webserver1 nach dem „reload“ des Loadbalancers eine erlaubte Aktivitätenabfolge ist. Dass die Abfolge jedoch nur nach einer entsprechenden Konfiguration des Loadbalancers, wenn keine Anfragen mehr an Webserver1 übermittelt werden, sinnvoll ist, wird in keinem Pattern berücksichtigt. Würde der „reload“ des Loadbalancers im Wartungsplan nicht doppelt vorkommen, würde dieser Fehler erkannt werden. Tabelle 8.8 stellt die Metriken Precision, Recall, F-Score und IG für das zweite Experiment dar.

Tabelle 8.8: Metriken für Experiment 2

Versuchsdurchlauf	Precision	Recall	F-Score	IG
1-20	0,361	0,951	0,523	0,277
1-10	0,48	0,923	0,632	0,238
11-20	0,259	1	0,411	0,375

Im Vergleich zu Experiment 1 fällt die deutlich gesunkene Precision und der Informationsgehalt auf. Dies ist hauptsächlich auf die doppelten Aktivitäten zurückzuführen. Der Informationsgehalt lässt sich durch das Herausfiltern von doppelten Fehlermeldungen leicht verbessern. Allerdings ist dabei generell zu beachten, dass auch mehrfache Fehlermeldungen ihren Sinn haben können, wenn bewusst so lange auf einen Fehler hingewiesen werden soll, bis dieser behoben ist.

8.3.3 Ergebnisse Experiment 3

Experiment 3 wurde durchgeführt, um zu prüfen, ob die Auswertung der Anti-Pattern online während der Wartungsdurchführung erfolgen kann. Dazu wurde in mehreren Versuchsdurchläufen die Anzahl der auf einmal versendeten Vorgang-Ereignisse, sowie die Anzahl der zu überprüfenden Wartungspläne schrittweise erhöht. In Tabelle 8.9 werden die einzelnen Testergebnisse übersichtlich dargestellt. Für jeden Testlauf, z.B. mit einem Wartungsplan und 10 gleichzeitig gestarteten Threads, wird die Anzahl der insgesamt erzeugten Fehlermeldungen, sowie die durchschnittliche Auswertungszeit je Fehlermeldung bzw. je Anti-Pattern dargestellt. Nachfolgend werden die Ergebnisse im Detail beschrieben.

Tabelle 8.9: Ergebnisse für Experiment 3

	1 Wartungsplan/ 17 Aktivitäten			10 Wartungspläne/ 170 Aktivitäten			100 Wartungspläne/ 1.700 Aktivitäten		
Anzahl Threads	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**
Anzahl Fehlermeldungen	352	3.944	39.950	2.860	29.080	583.754	28.020	289.788	-
Ø Auswertungs-zeit in ms	1,2	2,1	15,4	0,4	1,0	4,1	0,2	1,0	-

Insgesamt wurde eine durschnittliche Auswertungszeit je Vorgang-Ereignis von maximal 15,4 Millisekunden gemessen. Somit wird die Anforderung nach einer Erkennung von Wartungsfehlern online während der Wartungsdurchführung erfüllt. Bei der Interpretation der Messergebnisse fällt auf, dass die Anzahl an Fehlermeldungen sehr hoch ist. Dies ist darauf zurückzuführen, dass zum einen bewusst der Wartungsplan aus Experiment 2 für dieses Experiment ausgewählt wurde, um viele Fehlermeldungen zu erzeugen. Zum anderen werden in jedem Thread dieselben Ereignisse versendet, was dazu führt, dass z.B. das Anti-Pattern ABWESENHEIT beim zweiten Thread für jede Aktivität meldet, dass diese zu häufig ausgeführt wurde. Ein solches Szenario ist nicht sehr praxisnah. Für die Messung der Auswertungszeit je Fehlermeldung erfüllt es jedoch seinen Zweck.

Bei einer Wiederholung der Testfälle ist aufgefallen, dass die Anzahl an Fehlermeldungen leicht variieren kann. Vermutlich ändert sich die Reihenfolge der versendeten Vorgangereignisse in SoapUI, wodurch sich die erzeugten Fehlermeldungen ändern. Die Auswertungszeiten je Fehlermeldung sind jedoch bei einer Wiederholung der Testläufe weitgehend konstant geblieben.

Der Testlauf mit 100 Wartungsplänen und 1000 Threads wurde nach über 4 Stunden abgebrochen, weswegen für diesen Testlauf in Tabelle 8.9 keine Werte aufgeführt sind. In diesem Testlauf wurden gleichzeitig 13.000 Vorgang-Ereignisse an die Prüfeinheit gesendet. Die gleichzeitige Auswertung so vieler Vorgang-Ereignisse ist in der betrieblichen Praxis als sehr unwahrscheinlich einzustufen, zeigt jedoch die Grenzen der Methode auf.

Die Anzahl der Wartungspläne und Aktivitäten in einem Wartungsplan war für die Auswertungszeit je Fehlermeldung unkritisch, da die durchschnittliche Auswertungszeit mit steigender Anzahl an Wartungsplänen sogar abgenommen hat. Offensichtlich ist bei der Berechnung eines Anti-Patterns das Laden der betroffenen Aktivitäten aus den In Memory Event Tables am rechenintensivsten und die anschließende weitere Auswertung der Anti-

Pattern und somit die Erzeugung der Fehlermeldungen weniger rechenintensiv. Da bei 100 Wartungsplänen mehr Aktivitäten geladen und Fehlermeldungen für ein Vorgang-Ereignis erzeugt werden, verteilt sich das rechenintensivere Laden der Aktivitäten aus den Event Tables auf eine größere Anzahl erzeugter Fehlermeldungen.

8.4 Zusammenfassung des Kapitels

In diesem Kapitel wurden die Evaluationsergebnisse vorgestellt. Es wurden insgesamt drei Experimente durchgeführt. Zwei Experimente dienten der Überprüfung der Fehlererkennungsquote der Methode. Dazu wurden zwei verschiedene IT-Service-Wartungen ausgeführt und während der Wartungen typische Fehler erzeugt. Die Ergebnisse wurden durch die Metriken Recall, Precision, F-Score und Informationsgehalt gemessen. Eine Stärke der Methode liegt darin, dass der Wert Recall sehr hoch ist, d.h. es wurden bis auf zwei Situationen alle durchgeführten Fehler erkannt. Die Metriken Precision und Informationsgehalt bieten jedoch noch Potenzial für Verbesserungen. Die Metrik Precision kann durch explizit modellierte Fehlerbehandlungen in den Wartungsplänen verbessert werden. Der Nachteil davon ist jedoch, dass Wartungspläne sehr komplex werden können und bei einer manuellen Erstellung der Aufwand für die Erstellung von Wartungsplänen somit deutlich steigen dürfte. Ein alternativer Ansatz zur Verbesserung der Metrik Precision könnte der Einsatz von maschinellen Lern-Verfahren sein, um z.B. zu erkennen, welche Aktivitäten vermutlich aufgrund einer vorhergehenden Fehlerbehandlung durchgeführt wurden.

Die Metrik Informationsgehalt lässt sich durch eine Filterung der ausgehenden Fehlermeldungen verbessern. Eine hohe Anzahl an Fehlermeldungen kann zwei Gründe haben. Ein Grund kann sein, dass ein Fehler von mehreren Anti-Pattern gleichzeitig entdeckt wird, z.B. wenn vergessen wurde, eine Aktivität durchzuführen. Zur Veranschaulichung soll Abbildung 8.4 dienen. Wenn vergessen wurde, Aktivität b durchzuführen, dann würden die Anti-Pattern NACHFOLGER, VORGÄNGER und ANWESENHEIT jeweils eine Fehlermeldung für denselben Fehlergrund erzeugen.

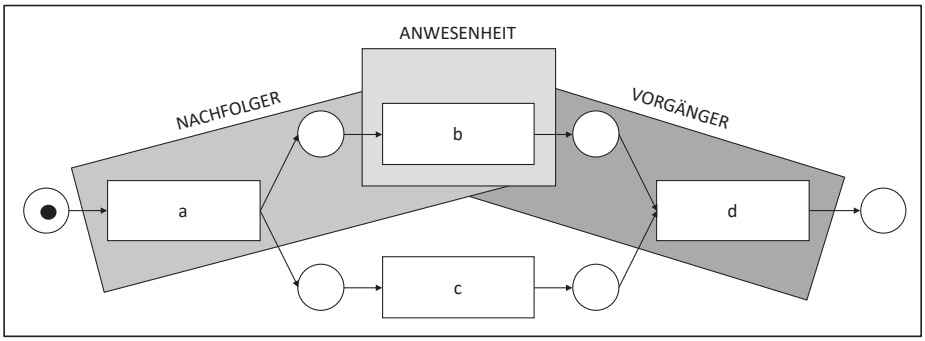


Abbildung 8.4: Beispiel für mehrfache Fehlermeldungen

Um die Anzahl an Fehlermeldungen in solchen Situationen zu verringern, kann bei einem betrieblichen Einsatz ein zusätzlicher Filter in der Prüfeinheit verwendet werden, der bei mehrfachen Fehlermeldungen mit demselben Fehlergrund nur eine Fehlermeldung anzeigt.

Ein weiterer Grund für mehrfache Fehlermeldungen ist, dass ein Fehler nicht behoben wird. Wenn beispielsweise in Abbildung 8.4 die Aktivität b nicht durchgeführt wird, dann erzeugt das Anti-Pattern NACHFOLGER bei jeder durchgeführten Aktivität, die nach der Aktivität b ausgeführt werden muss, eine Fehlermeldung, dass zunächst Aktivität b ausgeführt werden muss. Bei diesem Fall macht die mehrfache Erzeugung von Fehlermeldungen mit demselben Fehlergrund Sinn, damit der IT-Service-Anbieter daran erinnert wird, den Fehler zu beheben. Um kenntlich zu machen, dass es sich jedoch um denselben Fehlergrund handelt, könnte die Fehlermeldung entsprechend gekennzeichnet werden. Dazu kann ebenfalls ein Filter in der Prüfeinheit erstellt werden, der alle Fehlermeldungen des Anti-Patterns NACHFOLGER auf denselben Fehlergrund überprüft und entsprechend markiert.

Als drittes Experiment wurde die Performance der Auswertung der Anti-Pattern gemessen. Die durchschnittliche Auswertungszeit eines Anti-Patterns lag im niedrigen Millisekunden-Bereich, wodurch die Anforderung nach einer Online-Auswertung mit dieser Methode erfüllt wird. Der Prototyp kam erst bei der Auswertung von 13.000 gleichzeitig auszuwertenden Vorgang-Ereignissen und 1.700 Aktivitäten an seine Grenzen. Eine solch hohe Anzahl an gleichzeitig auszuwertenden Vorgang-Ereignissen ist bei der Wartung im betrieblichen Einsatz jedoch sehr unwahrscheinlich. Bei einer manuellen oder (teil)-automatisierten IT-Service-Wartung ist die Anzahl an gleichzeitig erzeugten Vorgang-Ereignissen naturgemäß deutlich geringer. Selbst bei einer automatisierten IT-Service-Wartung, beispielsweise bei einem sogenannten Rolling Upgrade (siehe [DuNa09]), bei dem ein Server nach dem anderen upgegradet wird, werden die Vorgang-Ereignisse sequenziell erzeugt (z.B. Stoppen eines Servers, Installieren einer neuen Softwareversion, Starten des

Servers), wodurch die Anzahl gleichzeitig auszuwertender Vorgang-Ereignisse wesentlich geringer ist. Um bei einem Rolling Upgrade dieselbe Last zu erzeugen wie im letzten Testlauf des Experiments, müssten z.B. 13.000 Server gleichzeitig gestoppt werden. Im nachfolgenden Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst, diskutiert und Beschränkungen der Methode dokumentiert.

9 Zusammenfassung, Beschränkungen und Ausblick

Nachfolgend werden die wesentlichen Ergebnisse dieser Arbeit zusammengefasst. Anschließend werden die methodischen sowie praktischen Beschränkungen der in dieser Arbeit entwickelten Konzepte dokumentiert und abschließend ein Ausblick auf weiterführende Forschungsaktivitäten gegeben.

9.1 Zusammenfassung

Einer der Hauptgründe für IT-Service-Ausfälle sind menschliche Fehler bei der Durchführung von IT-Service-Wartungen. Im Rahmen dieser Arbeit wurde eine Methode entwickelt, um Fehler bei der Durchführung einer IT-Service-Wartung direkt während der Durchführung online zu erkennen. Dadurch soll es dem IT-Service-Anbieter möglich sein, die erkannten Fehler noch während der eigentlichen IT-Service-Wartung zu beheben, um einen IT-Service-Ausfall nach Beendigung der IT-Service-Wartung aufgrund unerkannter und nicht behobener Durchführungsfehler zu vermeiden.

Zunächst wurden die Grundlagen des IT-Service-Managements dokumentiert, welches Prozesse für das Management von IT-Services umfasst. Dazu wurden zu Beginn die Begriffe Service und IT-Service voneinander abgegrenzt und definiert, sowie die an einem IT-Service beteiligten Rollen beschrieben. Einer der Prozesse des IT-Service-Managements ist der Prozess Change Management, der sich explizit mit der Wartung von IT-Services befasst. Dieser Prozess wurde detailliert beschrieben, da er organisatorisch als übergeordneter Rahmen für die Anwendung der in dieser Arbeit vorgestellten Methode dient. Ein zentrales Werkzeug des Change Managements ist der Change Plan. Im Change Plan werden die Aktivitäten zur Durchführung einer IT-Service-Wartung beschrieben.

Aufbauend auf den Grundlagen des IT-Service-Managements wurden in Kapitel 3 mögliche Fehler, die zu einer IT-Service-Störung führen können, klassifiziert und das allgemeine Prinzip der Fehlererkennung beschrieben. Die Klassifikation von Fehlern in IT-Services erfolgt nach den neun Dimensionen Betrachtungseinheit, Phänomenologischer Grund, IT-Service-Phase, IT-Service-Sicht, Ziel, Kenntnis, Fähigkeit, Persistenz und Entstehungsprozess. Die in dieser Arbeit entwickelte Methode eignet sich für die Entdeckung von Softwarewartungsfehlern, die bei der Umsetzung des Change Plans durch den IT-Service-Anbieter auftreten

können. Die Fehler werden von außen in den IT-Service eingebracht. Beim Einsatz der Methode ist es unerheblich, ob die Fehler aufgrund mangelnder Fähigkeiten der durchführenden Personen der IT-Service-Wartung eingespielt werden, und ob die Fehler von den durchführenden Personen der IT-Service-Wartung selbst bemerkt werden oder nicht. Die Methode ist zudem unabhängig von der Persistenz des Fehlers im IT-Service anwendbar. Wartungsfehler können unterschieden werden in Ablauffehler und Konfigurationsfehler. Eine Methode zur Entdeckung von Wartungsfehlern muss somit in der Lage sein, sowohl Ablauffehler als auch Konfigurationsfehler zu erkennen.

Das allgemeine Prinzip der Fehlererkennung besagt, dass zur Erkennung von Fehlern mindestens zwei Werte von einer Prüfeinheit auf einen Zusammenhang zwischen diesen Werten geprüft werden. In dieser Arbeit werden Vorgang-Ereignisse und Zustand-Ereignisse gewarteter IT-Service-Komponenten gegen Pattern-Instanzen eines vorab modellierten Wartungsplans geprüft. Die Prüfung des Zusammenhangs zwischen diesen Werten erfolgt in der Prüfeinheit mithilfe von Anti-Pattern. Wenn die Prüfeinheit eine Anti-Pattern-Instanz erkennt, wird eine entsprechende Fehlermeldung erzeugt.

Die Modellierung eines Wartungsplans wurde in Kapitel 4 beschrieben. Zunächst wurden die Grundlagen zur Modellierung von Abläufen dokumentiert. Als Modellierungsmethode wurden dabei insbesondere Petri-Netze, einfache Petri-Netze und höhere Petri-Netze beschrieben. Petri-Netze haben im Vergleich zu anderen Modellierungssprachen wie EPKs oder BPMN den Vorteil, dass sie formalisiert sind und dadurch vielfältige Analysemethoden unterstützen. Eine Variante höherer Petri-Netze bilden XML-Netze. Diese wurden im Detail beschrieben, da sie für die Modellierung von Wartungsplänen zum Einsatz kommen. In einem XML-Netz repräsentieren Stellen Behälter für XML-Dokumente. XML erlaubt die Modellierung komplex strukturierter Objekte wie IT-Service-Komponenten. Ein Standard, der die Modellierung von IT-Service-Komponenten in Form von XML ermöglicht, ist TOSCA. Zum Schluss des vierten Kapitels wurde beschrieben, wie mithilfe eines angepassten TOSCA XML Schemas und XML-Netzen IT-Service-Wartungen modelliert werden können. Die daraus entstehenden Modelle werden in dieser Arbeit Wartungspläne genannt. Wartungspläne bieten den Vorteil, dass sie die Modellierung des Ablaufs einer IT-Service-Wartung und die Struktur der gewarteten IT-Service Komponenten in einem Modell integrieren. Das ist wichtig, um aus einem Wartungsplan Pattern-Instanzen ableiten zu können, die es ermöglichen, sowohl Ablauf- als auch Konfigurationsfehler zu erkennen.

Nachdem die Modellierung von IT-Service-Wartungen in Form von Wartungsplänen beschrieben wurde, wurde in Kapitel 5 dokumentiert, welche Pattern und Anti-Pattern zum Einsatz kommen, um Wartungsfehler zu erkennen. Hierzu wurden zunächst die Begriffe

Pattern und Anti-Pattern definiert, sowie ein Überblick über bekannte Pattern im Umfeld von betrieblichen Prozessen gegeben. Anschließend wurde der konkrete Kontext für diese Arbeit definiert, um daraufhin die Pattern und Anti-Pattern für diesen Kontext zu beschreiben. Insgesamt werden die zehn Pattern/Anti-Pattern ANWESENHEIT, ABWESENHEIT, ALTERNATIVE ANWESENHEIT, ALTERNATIVE ABWESENHEIT, NACHFOLGER, DIREKTER NACHFOLGER, VORGÄNGER, DIREKTER VORGÄNGER, WERT und ZUSTAND definiert.

In Kapitel 6 wurde das Vorgehen zum Einsatz der in dieser Arbeit entwickelten Methode zur Entdeckung von Wartungsfehlern im Rahmen des IT-Service-Managements dokumentiert. Dazu wurde zunächst die Erstellung des Wartungsplans in den Prozess des Change Managements, der in Kapitel 2 beschrieben wurde, eingeordnet. Anschließend wurden mögliche Analysen von Wartungsplänen beschrieben, die verwendet werden können, um bereits vor der Durchführung einer IT-Service-Wartung zu prüfen, ob der Wartungsplan überhaupt erfolgreich durchgeführt werden kann. Ausgehend von der Erstellung des Wartungsplans wurde daraufhin dokumentiert, wie Pattern-Instanzen aus einem Wartungsplan abgeleitet werden können. Zunächst wird dafür ein Wartungsplan simuliert, um ein Ereignis-Log zu erhalten. Dieses Ereignis-Log wird zur Erstellung eines sogenannten Fußabdrucks des Wartungsplans verwendet. Aus dem Fußabdruck eines Wartungsplans können wiederum die Pattern des Wartungsplans instanziiert werden. Es wurde für jedes in Kapitel 5 beschriebene Pattern dokumentiert, wie dieses instanziiert werden kann. Die Erzeugung eines Fußabdrucks ist eine Methode, die beim Process Mining verwendet wird. Daher wurden in Kapitel 6 kurz die Grundlagen des Process Minings, insbesondere der beiden Anwendungsfelder „Entdeckung“ und „Übereinstimmung“ dokumentiert.

Um die Methode evaluieren zu können, wurde ein Software-Prototyp erstellt, der in Kapitel 7 vorgestellt wurde. Zunächst werden Anforderungen an den Prototyp beschrieben, die sich aus der Motivation für die Arbeit in Kapitel 1 ergeben. Anschließend wurde eine Architektur vorgeschlagen und diese im Detail beschrieben. Die vorgeschlagene Architektur besteht aus fünf Hauptkomponenten, der Modellierungskomponente, der Überwachungsanwendung, den Log-Agenten, dem Log-Filter und der Prüfeinheit. Die Modellierungskomponente dient zur Erstellung eines Wartungsplans und der Instanziierung von Pattern-Instanzen. Diese werden an die Prüfeinheit übertragen. Da die Prüfeinheit Pattern-Instanzen gegen Vorgang-Ereignisse und Zustand-Ereignisse prüft, werden Komponenten benötigt, die diese Ereignisse an die Prüfeinheit senden. Log-Agenten extrahieren dazu Vorgang-Ereignisse aus den Logs der zu überwachenden IT-Service-Komponenten und senden diese an einen Log-Filter. Im Log-Filter werden die Vorgang-Ereignisse standardisiert und anschließend an die Prüfeinheit gesendet. Die Prüfeinheit wird in dieser Arbeit mithilfe einer Complex Event Processing Engine realisiert. Daher wurden in dem Kapitel zusätzlich die Grundlagen des

Complex Event Processings und der Sprache SiddhiQL beschrieben. SiddhiQL wird zur Beschreibung von Ereignisregeln genutzt. Damit eine Complex Event Processing Engine die in Kapitel 5.4 vorgestellten Anti-Pattern prüfen kann, müssen diese in Form von Ereignisregeln beschrieben werden. Dazu wurden exemplarisch die beiden Anti-Pattern NACHFOLGER und DIREKTER NACHFOLGER als SiddhiQL-Ereignisregeln formuliert und detailliert erklärt. Die weiteren als SiddhiQL-Ereignisregeln definierten Anti-Pattern können Anhang A entnommen werden. Zum Schluss des Kapitels wurde eine prototypische Implementierung der vorgestellten Architektur beschrieben, die zur Evaluation der Methode verwendet wurde.

In Kapitel 8 wurde die Evaluation der Methode vorgestellt. Dazu wurden insgesamt drei verschiedene Experimente durchgeführt. Zu Beginn des Kapitels wurde zunächst der Versuchsaufbau für die Experimente beschrieben. Um die Methode zu evaluieren, wurde als Versuchsaufbau eine Umgebung bei Amazon Web Services mit insgesamt acht Maschineninstanzen erstellt, auf denen als IT-Service das Customer Relationship Management System SugarCRM installiert wurde. Mit den ersten beiden Experimenten wurde die Erkennungsrate von Fehlern geprüft. Dazu wurden zwei jeweils unterschiedliche Wartungspläne erstellt und absichtlich Fehler während der Wartung durchgeführt, um zu prüfen, ob beim Einsatz der Methode die injizierten Fehler entdeckt werden. Beim ersten Experiment wurde dabei der Fokus im Wartungsplan auf nebenläufige Aktivitäten gelegt. Der Recall lag bei diesem Experiment bei 1. Die Precision lag bei den ersten Versuchsdurchläufen, bei denen erkannte Fehler nicht sofort korrigiert wurden, ebenfalls bei 1. Bei den Versuchsdurchläufen, bei denen die Fehler direkt nach ihrer Erkennung behoben wurden, sank die Precision auf 0,436. Das ist damit zu begründen, dass im Wartungsplan keine Fehlerbehandlung modelliert wurde und die Fehlerbehebungen somit als Abweichung vom Wartungsplan und damit als Fehler gewertet wurden. Im zweiten Experiment wurde der Fokus auf alternative Aktivitäten und doppelte Aktivitäten gelegt, um bewusst Schwachstellen der Methode zu identifizieren. Der Recall lag bei diesem Experiment bei 0,951 und die Precision insgesamt bei 0,361. Als Schwachstelle der Methode wurden bei diesem Experiment die doppelten Aktivitäten identifiziert, die zu falschen Fehlermeldungen führten und somit zu einer geringeren Precision als in Experiment 1. Außerdem wurden teilweise Fehlermeldungen mehrfach erzeugt, was zu einer sehr hohen Anzahl an Fehlermeldungen führte. Dadurch besteht die Gefahr, dass Fehlermeldungen vom IT-Service-Anbieter nicht mehr ernst genommen werden. In Experiment 3 wurde die Anforderung nach einer Online-Erkennung von Fehlern während der Durchführung einer IT-Service-Wartung evaluiert. Dazu wurden Wartungspläne mit 17 bis 1.700 Aktivitäten in die Prüfeinheit geladen und 130 bis 13.000 Vorgang-Ereignisse auf einmal an die Prüfeinheit gesendet. Die Auswertungszeit der Prüfeinheit je Fehlermeldung belief sich dabei auf 0,2 bis 15,4 Millisekunden. Die Anforderung nach einer

Online-Erkennung von Fehlern während einer IT-Service-Wartung kann mit der Methode somit erfüllt werden. Nachdem die Ergebnisse dieser Arbeit zusammengefasst wurden, werden nachfolgend Herausforderungen und Beschränkungen der vorgestellten Methode dokumentiert.

9.2 Herausforderungen und Beschränkungen

Als Herausforderungen werden nachfolgend ungelöste Probleme verstanden, die lösbar sind. Als Beschränkungen werden hingegen Aspekte verstanden, die methodische Grenzen aufzeigen. Bei den Herausforderungen werden nicht standardisierte Logs und Konfigurationsdateien, sowie Fehlermeldungen aufgrund nicht modellierter Fehlerbehandlungen beschrieben. Bei den Beschränkungen werden doppelte Aktivitäten und die Überprüfung der Wartung von Konfigurationsparametern beschrieben.

9.2.1 Nicht standardisierte Logs & Konfigurationsdateien

Eine Herausforderung beim praktischen Einsatz der Methode sind nicht standardisierte Logs und Konfigurationsdateien. Als Beispiel sei das Betriebssystem Debian[1] genannt, das verschiedene Konfigurationsdateien enthält, die sich von der Syntax her voneinander unterscheiden [ŚwBo17]. Ebenso können sich Log-Dateien vom Format her zwischen verschiedenen Anwendungen voneinander unterscheiden. Das führt dazu, dass die benötigten Log-Agenten und der Log-Filter entsprechend für jedes neue Format von Konfigurationsdateien oder Log-Dateien angepasst bzw. erweitert werden müssen. Die Formate von Log-Dateien können sich selbst innerhalb einer einzelnen Anwendung unterscheiden, z.B. können verschiedene Formate von Zeitstempeln verwendet werden, abhängig davon, auf welchem Betriebssystem die Anwendung installiert wurde. Dies war z.B. während der Evaluation beim MySQL-Server der Fall, der abhängig vom Betriebssystem, unterschiedliche Zeitstempelformate in den Log-Dateien verwendete. Zudem kann es bei Anwendungen vorkommen, dass Konfigurationsparameter nicht eindeutig sind und ein Konfigurationsparameter mehrfach vorkommen kann. Dies ist beispielsweise beim Loadbalancer nginx beim Konfigurationsparameter „server“ der Fall.

Eine weitere Herausforderung bei der Auswertung von Log-Dateien ist das Log-Level. Das Log-Level einer Anwendung bestimmt, in welchem Detaillierungsgrad Einträge in das Log geschrieben werden. Um zur Anwendung der Methode alle benötigten Informationen in einem Log zu erhalten, kann es notwendig sein, ein hohes Log-Level zu verwenden. Das

[1] https://www.debian.org

führt dazu, dass sehr viele Einträge in ein Log geschrieben werden und somit die manuelle Analyse von Logs, z.B. durch Systemadministratoren, erschwert werden kann. Eine weitere Auswirkung durch das Erhöhen des Log-Levels ist der zusätzlich benötigte Speicherplatz aufgrund der dadurch steigenden Dateigröße. Dies ist beim Betrieb eines IT-Services zu berücksichtigen, so dass entweder zusätzlicher Festplattenspeicher benötigt wird, oder die Log-Dateien regelmäßig gelöscht bzw. archiviert werden müssen.

Eine zusätzliche Herausforderung bei der Auswertung von Log-Dateien besteht darin, dass nicht jede Aktivität direkt aus einem Log gelesen werden kann. Beispielsweise wird beim Loadbalancer nginx der reload-Befehl nicht als solcher in das Log geschrieben. Stattdessen werden in das Log dieselben Einträge geschrieben wie bei einem manuellen Stoppen und Starten des Loadbalancers. Um trotzdem Aktivitäten eindeutig in einem Log zu identifizieren, können Cluster von Log-Einträgen gebildet werden, die wiederum eine Aktivität repräsentieren (siehe [FSWG17]).

9.2.2 Fehler aufgrund nicht modellierter Fehlerbehandlung

Wenn durch den Einsatz der Methode ein Fehler während einer IT-Service-Wartung erkannt wird, ist davon auszugehen, dass dieser auch direkt durch den IT-Service-Anbieter behoben wird. Dies stellt eine Herausforderung dar, wenn diese Fehlerbehandlung nicht explizit im Wartungsplan modelliert ist. Dann meldet die Methode nämlich eine fehlerhafte Ausführung des Wartungsplans, obwohl die Fehlerbehandlung in diesem Fall sinnvoll und richtig gewesen wäre. Die explizite Modellierung aller möglichen Fehlerbehandlungen ist jedoch nicht zielführend, da der Wartungsplan dadurch sehr komplex werden kann. Die Modellierung von Fehlerbehandlungen eignet sich für schwerwiegende Fehler, die nicht intuitiv korrigiert werden können, wie beispielsweise, wenn vergessen wurde, einen Server zu starten, oder wenn ein Rollback der IT-Service-Wartung notwendig wird und somit sogar ein anderer Zielzustand als ursprünglich geplant erreicht wird.

Für den Umgang mit „einfachen“ Fehlern ist die explizite Modellierung von Fehlerbehandlungen nicht praxistauglich. Ideen und Ansätze zur Lösung dieses Problems können eventuell Arbeiten zum Thema „Concept Drift“ in den Forschungsbereichen Data Mining [GŽBP14] und Process Mining [BAŽP14] liefern. In diesem Themenbereich wird versucht, Änderungen des Daten-/Ablauf-Modells während der Datenanalyse zu lokalisieren, zu charakterisieren und anschließend im Modell und der laufenden Datenanalyse zu berücksichtigen [BAŽP14]. In der Domäne dieser Arbeit ist eine Änderung des Wartungsplans während der IT-Service-Wartung unwahrscheinlich, da es sich um sehr kurzlebige Prozesse handelt. Nichtsdestotrotz stellt die Fehlerbehebung eines einfachen Fehlers, die nicht explizit modelliert wurde,

eine Änderung von der geplanten Durchführung einer IT-Service-Wartung dar und sollte als solche erkannt und bei der Erzeugung von Fehlermeldungen entsprechend berücksichtigt werden. Die Lokalisation der Abweichung vom Wartungsplan kann bereits durch die erzeugte Fehlermeldung erfolgen, z.B. wenn vergessen wurde, eine Aktivität durchzuführen. Für die Charakterisierung und Entscheidung, ob es sich um einen tatsächlichen Fehler oder eine Fehlerbehandlung handelt, ist dies jedoch nicht ausreichend und bleibt somit eine offene Fragestellung, die weiter untersucht werden muss.

9.2.3 Doppelte Aktivitäten

Eine methodische Beschränkung sind doppelte Aktivitäten. Als doppelte Aktivität werden zwei Aktivitäten in einem Wartungsplan verstanden, die in den Attributen st, *app*, *op*, *prop* und *value* identisch sind. Eine Aktivität, die im Rahmen eines Zyklus mehrfach ausgeführt werden kann, ist keine doppelte Aktivität. Bei doppelten Aktivitäten kann es vorkommen, dass entweder eine falsche Fehlermeldung erzeugt wird oder ein Fehler nicht erkannt wird.

Die erste Situation, in der eine falsche Fehlermeldung erzeugt wird, betrifft das Anti-Pattern NACHFOLGER. Dieser Fall tritt beispielsweise im in Kapitel 8.2.2 vorgestellten Experiment auf und wird beispielhaft auch in Abbildung 9.1 dargestellt. Damit es zu falschen Fehlermeldungen kommt, kann die doppelte Aktivität selbst nebenläufig ausgeführt werden oder die nach der doppelten Aktivität auszuführende Aktivität kann nebenläufig ausgeführt werden. Um die Ursache für die falschen Fehlermeldungen zu erklären, wurden die Transitionen in Abbildung 9.1 mit den Buchstaben a-f markiert. In Abbildung 9.1 kommt die Aktivität „reload“ bzw. a doppelt vor. Die Pattern-Instanzen für das Pattern NACHFOLGER mit der Aktivität a sind (a, b, d), (a, b, e), (a, b, f), (a, c, d), (a, c, e) und (a, c, f). Das Anti-Pattern NACHFOLGER erzeugt falsche Fehlermeldungen, wenn nach der Aktivität a die Aktivitäten e und f durchgeführt werden. Der Grund dafür ist, dass gemäß den Pattern-Instanzen (a, b, e), (a, b, f), (a, c, e) und (a, c, f) zwischen der Durchführung der Aktivitäten a und e bzw. f die Aktivitäten b und c durchgeführt sein müssen. Dies ist nur beim ersten Auftreten von a der Fall, nicht aber beim zweiten Auftreten von a, weshalb das Anti-Pattern eine falsche Fehlermeldung erzeugt.

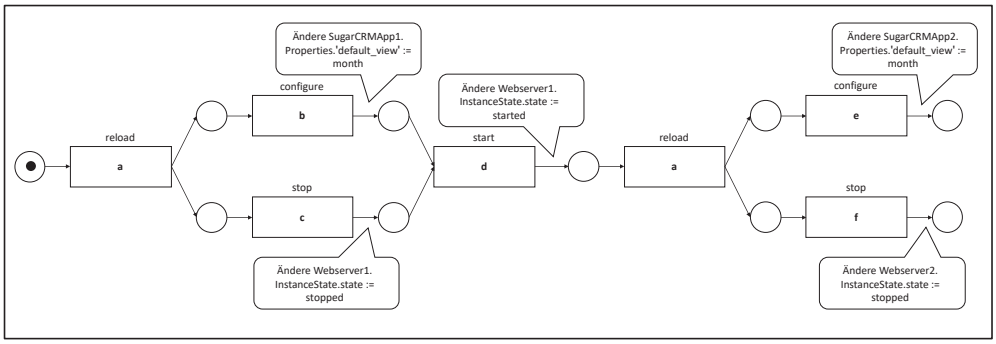


Abbildung 9.1: Beispiel für falsche Fehlermeldungen

Die zweite Situation, in der eine falsche Fehlermeldung erzeugt wird, betrifft die Anti-Pattern VORGÄNGER und DIREKTER VORGÄNGER. Wenn eine doppelte Aktivität die erste Aktivität in einem Wartungsplan ist, würde eine Fehlermeldung erzeugt werden, da das entsprechende Anti-Pattern in der Vorgangereignishistorie prüft, ob bereits die Aktivität vor der zweiten doppelten Aktivität ausgeführt wurde. Da dies bei der ersten doppelten Aktivität nicht der Fall ist, würde eine falsche Fehlermeldung erzeugt werden. Dieser Fall tritt beispielsweise in Abbildung 9.1 beim Ausführen der ersten doppelten Aktivität a auf.

Die dritte Situation, in der ein Fehler nicht erkannt wird, betrifft doppelte Aktivitäten in einer nicht nebenläufigen Sequenz wie in Abbildung 9.2 abgebildet. In diesem Beispiel kommen die Anti-Pattern DIREKTER NACHFOLGER und DIREKTER VORGÄNGER zum Einsatz. Wenn während der Durchführung dieses Wartungsplans nun zunächst SugarCRMApp2 konfiguriert wird, anstatt SugarCRMApp1, würde dieser Fehler nicht erkannt werden, da bei diesen beiden Anti-Pattern immer nur die direkt nachfolgende bzw. vorausgehende Aktivität geprüft wird. Da nach der Aktivität „reload“ sowohl die Konfiguration von SugarCRMApp1, als auch von SugarCRMApp2 erlaubt ist, würde der Fehler unentdeckt bleiben. Dieselbe Situation kann beim Anti-Pattern VORGÄNGER bei nebenläufigen Sequenzen auftreten.

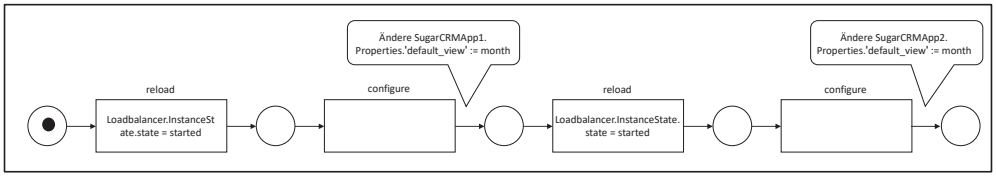


Abbildung 9.2: Beispiel für nicht erkannte Fehler

9.2.4 Konfigurationsparameter

Eine Fehlerart bei Konfigurationsfehlern sind syntaktische Fehler der Art, dass ein Konfigurationsparameter in einer Konfigurationsdatei vor einem anderen Parameter gesetzt sein muss, wie in Kapitel 3.3 in Tabelle 3.2 beschrieben. Solche Fehler können nur teilweise erkannt werden. Beim Einsatz der in dieser Arbeit vorgestellten Methode werden solche Fehler erkannt, wenn die Parameter zeitlich nacheinander gesetzt werden und diese zeitliche Abfolge auch der Reihenfolge der Parameter in der Konfigurationsdatei entspricht. Wenn beispielsweise ein Konfigurationsparameter a sowohl zeitlich als auch in der Konfigurationsdatei vor einem Konfigurationsparameter b gesetzt sein muss, und es wird bei der IT-Service-Wartung auch tatsächlich a zeitlich nach b gesetzt, dann würde keine Fehlermeldung erzeugt werden. Wurde b jedoch zwar zeitlich nach a gesetzt, in der Konfigurationsdatei jedoch trotzdem fälschlicherweise vor a geschrieben, dann würde dieser Fehler nicht erkannt werden. Die Methode könnte dahin verbessert werden, dass nicht nur ein einzelner Konfigurationsparameter geprüft wird, sondern der gesamte Pfad des Konfigurationsparameters. Aus dem Wartungsplan kann die Information des Pfades zu einem Konfigurationsparameter aufgrund der XML-Struktur extrahiert werden. Bei der zu überprüfenden Konfigurationsdatei kann es jedoch möglich sein, dass diese eine sehr flache Struktur mit nur einer einzigen Ebene hat. In diesem Fall wird auch mit der Pfadangabe des Konfigurationsparameters keine Aussage darüber möglich sein, ob der Konfigurationsparameter in der Datei korrekt gesetzt wurde. Die explizite Erkennung solcher strukturellen Fehler im Rahmen der in dieser Arbeit vorgestellten Methode ist somit noch ein offener Punkt.

9.3 Ausblick

Zusätzlich zu den aus methodischer Sicht offenen Punkten hinsichtlich doppelter Aktivitäten und der Prüfung struktureller Eigenschaften von Konfigurationsparametern ergeben sich noch folgende weitere Entwicklungs- und Forschugsmöglichkeiten.

Beim entwickelten Prototyp sollte die Simulationskomponente weiterentwickelt werden, so dass bei der Simulation von Wartungsabläufen die in Form von Marken hinterlegten TOSCA Service Templates berücksichtigt werden. Dies beinhaltet sowohl die Berücksichtigung der TOSCA Service Templates bei der Ablaufsteuerung, als auch die Simulation der Veränderung eines TOSCA Service Templates selbst. Dadurch kann das bei der Simulation erzeugte Service Template für weitere IT-Service-Wartungen verwendet werden, ohne dass dieses zuvor manuell aktualisiert werden muss.

Eine weitere Entwicklungsmöglichkeit bietet der Pattern-Instanzen Generator. Durch die Implementierung des Generators wird der Aufwand zur manuellen Erstellung der Pattern-Instanzen vermieden. Mithilfe der erweiterten Simulationskomponente und des Pattern-Instanzen Generators würde nur noch ein manueller Aufwand für die Erstellung der Wartungspläne entstehen. Damit ließe sich die Methode in der betrieblichen Praxis über einen längeren Zeitraum und unter Einbindung in den Change Management Prozess evaluieren.

Zusätzlich zu den beiden zuvor genannten Entwicklungsmöglichkeiten im erstellten Prototyp, sollte die praktische Anwendbarkeit der Methode weiter verbessert werden. Unter der Annahme, dass die Anzahl an IT-Service-Wartungen mit Entwicklungen wie Cloud Computing und DevOps weiter zunehmen wird, bietet insbesondere der Aspekt der schnellen Erzeugung von Wartungsplänen weiteres Entwicklungspotenzial. Mögliche weitere Entwicklungsarbeiten betreffen beispielsweise die Wiederverwendung von Wartungsplänen oder die (teil-)automatisierte Generierung von Wartungsplänen.

Zudem sollte generell die automatisierte Durchführung von IT-Service-Wartungen weiter untersucht werden. Die in dieser Arbeit vorgestellten Wartungspläne bieten aufgrund ihres Formalismus die Chance, als Basis zur Beschreibung von technologie- und plattformübergreifenden IT-Service-Wartungen zu dienen. Gleichzeitig können die erzeugten Wartungspläne durch die bestehenden Analyse-Möglichkeiten von Petri-Netzen auf Korrektheit überprüft werden, um beispielsweise Fehler in Wartungsplänen zu erkennen. Ein unerkannter Fehler in einem Wartungsplan könnte sich ansonsten bei einer automatisierten Wartungsdurchführung schnell in mehreren IT-Service-Komponenten gleichzeitig manifestieren.

Weitere Entwicklungsmöglichkeiten bestehen darin, dass nicht nur die Logs einer IT-Service-Komponente ausgewertet werden, sondern zusätzlich bereits die Befehle, die an eine IT-Service-Komponente gesendet werden, überprüft werden. Die in dieser Arbeit vorgestellte Prüfeinheit könnte hierfür genutzt werden, so dass Befehle an IT-Service-Komponenten von der Prüfeinheit überprüft, und entweder an die entsprechende IT-Service-Komponente weitergeleitet, oder bei einem Fehler abgewiesen werden.

Literaturverzeichnis

[Aals11] Aalst van der, W. M. P.: Process mining: discovery, conformance and enhancement of business processes: Springer Science & Business Media, 2011

[Aals98] Aalst van der, W. M. P.: The application of Petri nets to workflow management. In: Journal of circuits, systems, and computers Bd. 8 (1998), Nr. 1, S. 21–66

[AAMA11] Aalst van der, W. M. P.; Adriansyah, A.; Medeiros, A. K. A. D.; Arcieri, F.; Baier, T.; Blickle, T.; Bose, J. C.; Brand, P. V. D.; u. a.: Process Mining Manifesto. In: Business Process Management Workshops, Lecture Notes in Business Information Processing: Springer, Berlin, Heidelberg, 2011, S. 169–194

[AaWe04] Aalst van der, W.M.P.; Weijters, A.J.M.M.: Process mining: a research agenda. In: Computers in industry Bd. 53 (2004), S. 231–244

[AaWM04] Aalst van der, W. M. P.; Weijters, T.; Maruster, L.: Workflow mining: Discovering process models from event logs. In: IEEE Transactions on Knowledge and Data Engineering Bd. 16 (2004), Nr. 9, S. 1128–1142

[ABEE15] Awad, A.; Barnawi, A.; Elgammal, A.; Elshawi, R.; Almalaise, A.; Sakr, S.: Runtime Detection of Business Process Compliance Violations: An Approach Based on Anti Patterns. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, SAC '15. New York, NY, USA: ACM, 2015, S. 1203–1210

[AHKB03] Aalst van der, W. M. P.; Hofstede ter, A. H. M.; Kiepuszewski, B.; Barros, A. P.: Workflow Patterns. In: Distributed and Parallel Databases Bd. 14 (2003), Nr. 1, S. 5–51

[Alex79] Alexander, C.: The Timeless Way of Building: Oxford University Press, 1979

[ALRL04] Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. In: Dependable and Secure Computing, IEEE Transactions on Bd. 1 (2004), Nr. 1, S. 11–33

[Amaz11] Amazon Web Services, Inc.: Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region. URL https://aws.amazon.com/message/65648/. - abgerufen am 2018-11-05

[ATRD15] Aalst van der, W. M. P.; Hofstede ter, A. H. M.; Russell, N.; Dumas, M.; Wohed, P.: Workflow Patterns | Patterns. URL http://www.workflowpatterns.com/patterns/index.php. - abgerufen am 2015-08-28

[BaGO04] Baida, Z.; Gordijn, J.; Omelayenko, B.: A Shared Service Terminology for Online Service Provisioning. In: Proceedings of the 6th International Conference on Electronic Commerce, ICEC '04. New York, NY, USA: ACM, 2004, S. 1–10

[Bart10] Bartsch, C.: Modellierung und Simulation von IT-Dienstleistungsprozessen. Karlsruhe, KIT, Fakultät für Wirtschaftswissenschaften, Dissertation, 2010

[Baum97] Baumgarten, B.: Petri-Netze: Grundlagen und Anwendungen. 2. Aufl. Heidelberg; Berlin; Oxford: Spektrum Akademischer Verlag, 1997

[BAŽP14] Bose, R. P. J. C.; Aalst van der, W. M. P.; Žliobaitė, I.; Pechenizkiy, M.: Dealing With Concept Drifts in Process Mining. In: IEEE Transactions on Neural Networks and Learning Systems Bd. 25 (2014), Nr. 1, S. 154–171

[BBHK13] Binz, T.; Breitenbücher, U.; Haupt, F.; Kopp, O.; Leymann, F.; Nowak, A.; Wagner, S.: OpenTOSCA–
a runtime for TOSCA-based cloud applications. In: International Conference on Service-Oriented Computing: Springer, 2013, S. 692–695

[BBKL14] Binz, T.; Breitenbücher, U.; Kopp, O.; Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications. In: Advanced Web Services: Springer, New York, NY, 2014, S. 527–549

[BCSW15] Brogi, A.; Canciani, A.; Soldani, J.; Wang, P. W.; Moldt, D.: Modelling the behaviour of management operations in cloud-based applications. In: Proceedings of the International Workshop on Petri Nets and Software Engineering, PNSE. Bd. 15, 2015, S. 191–205

[BeKl14] Becker, M.; Klingner, S.: A Criteria Catalogue for Evaluating Business Process Pattern Approaches. In: Bider, I.; Gaaloul, K.; Krogstie, J.; Nurcan, S.; Proper, H. A.; Schmidt, R.; Soffer, P. (Hrsg.): Enterprise, Business-Process and Information Systems Modeling, Lecture Notes in Business Information Processing: Springer Berlin Heidelberg, 2014, S. 257–271

[BeZi14] Beims, M.; Ziegenbein, M.: IT-Service-Management in der Praxis mit ITIL®: Der Einsatz von ITIL® Edition 2011, ISO/IEC 20000:2011, COBIT® 5 und PRINCE2®: Carl Hanser Verlag GmbH & Co. KG, 2014

[BHHK08] Buhl, H. U.; Heinrich, B.; Henneberger, M.; Krammer, A.: Service science. In: Wirtschaftsinformatik Bd. 50 (2008), Nr. 1, S. 60–65

[Binz15] Binz, T.: Crawling von Enterprise Topologien zur automatisierten Migration von Anwendungen: eine Cloud-Perspektive. Stuttgart, Universitätsbibliothek der Universität Stuttgart, Dissertation, 2015

[BMNN05] Bianchini, R.; Martin, R. P.; Nagaraja, K.; Nguyen, T. D.; Oliveira, F.: Human-aware Computer System Design. In: Proceedings of the 10th Conference on Hot Topics in Operating Systems - Volume 10, HOTOS'05. Berkeley, CA, USA: USENIX Association, 2005

[BoJK08] Bon van, J.; Jong de, A.; Kolthof, A.: Service Transition basierend auf ITIL V3: Ein Management Guide. In: Zaltbommel: Van Haren Publ (2008)

[BöKr04] Böhmann, T.; Krcmar, H.: Grundlagen und Entwicklungstrends im IT-Servicemanagement. In: HMD-Praxis der Wirtschaftsinformatik Bd. 237 (2004), S. 7–21

[BrDu15] Bruns, R.; Dunkel, J.: Complex Event Processing: Komplexe Analyse von massiven Datenströmen mit CEP, essentials: Springer Vieweg, 2015

[Brei16] Breitenbücher, U.: Eine musterbasierte Methode zur Automatisierung des Anwendungsmanagements. Stuttgart, Universitätsbibliothek der Universität Stuttgart, Dissertation, 2016

[Bren07] Brenner, M.: Werkzeugunterstützung für ITIL-orientiertes Dienstmanagement: ein modellbasierter Ansatz. Norderstedt: Books on Demand GmbH, 2007

[BrSW14] Brogi, A.; Soldani, J.; Wang, P. W.: TOSCA in a Nutshell: Promises and Perspectives. In: Service-Oriented and Cloud Computing, Lecture Notes in Computer Science: Springer, Berlin, Heidelberg, 2014, S. 171–186

[BuSc06] Bullinger, H.-J.; Schreiner, P.: Service Engineering: Ein Rahmenkonzept für die systematische Entwicklung von Dienstleistungen. In: Service Engineering: Springer, Berlin, Heidelberg, 2006, S. 53–84

[BuSt06] Burr, W.; Stephan, M.: Dienstleistungsmanagement: innovative Wertschöpfungskonzepte im Dienstleistungssektor. Stuttgart: Kohlhammer, 2006

[BWGG10] Bianchin, L. A.; Wickboldt, J. A.; Granville, L. Z.; Gaspary, L. P.; Bartolini, C.; Rahmouni, M.: Similarity metric for risk assessment in IT change plans. In: 2010 International Conference on Network and Service Management, 2010, S. 25–32

[By05] By, R. T.: Organisational change management: A critical review. In: Journal of Change Management Bd. 5 (2005), Nr. 4, S. 369–380

[ChSp06] Chesbrough, H.; Spohrer, J.: A Research Manifesto for Services Science. In: Commun. ACM Bd. 49 (2006), Nr. 7, S. 35–40

[Cisc08] Cisco Systems, Inc: Change Management: Best Practices. URL https://de.scribd.com/document/ 226086771/sicsco-change-management-best-practice-white-paper#. - abgerufen am 2018-11-03

[CLOS09] Che, H.; Li, Y.; Oberweis, A.; Stucky, W.: Web Service Composition Based on XML Nets. In: 42nd Hawaii International Conference on System Sciences, 2009. HICSS ’09, 2009, S. 1–10

[CMAS09] Cordeiro, W. L. D. C.; Machado, G. S.; Andreis, F. G.; Santos do, A. D.; Both, C. B.; Gaspary, L. P.; Granville, L Z; Bartolini, C.; u. a.: ChangeLedge: Change design and planning in networked systems based on reuse of knowledge and automation. In: Computer Networks Bd. 53 (2009), Nr. 16, S. 2782–2799

[CMAW09] Cordeiro, W. L. D. C.; Machado, G. S.; Andreis, F. G.; Wickboldt, J. A.; Lunardi, R. C.; Santos dos, A. D.; Both, C. B.; Gaspary, L. P.; u. a.: CHANGEMINER: A solution for discovering IT change templates from past execution traces. In: 2009 IFIP/IEEE International Symposium on Integrated Network Management, 2009, S. 97–104

[CoWE08] Conger, S.; Winniford, M. A.; Erickson-Harris, L.: Service Management in Operations. In: AMCIS 2008 Proceedings (2008)

[DeKL98] Deswarte, Y.; Kanoun, K.; Laprie, J.-C.: Diversity against accidental and deliberate faults. In: Proceedings Computer Security, Dependability, and Assurance: From Needs to Solutions, 1998, S. 171–181

[DeOb96] Desel, J.; Oberweis, A.: Petri-Netze in der Angewandten Informatik: Einführung, Grundlagen und Perspektiven: Anwendungen von Petri-Netzen. In: Wirtschaftsinformatik Bd. 38 (1996), Nr. 4, S. 359–367

[Dist12a] Distributed Management Task Force, Inc.: CIM Infrastructure Specification 2.7.0. URL https://www.dmtf.org/sites/default/files/standards/documents/DSP0004_2.7.0.pdf. - abgerufen am 2016-09-03

[Dist12b] Distributed Management Task Force, Inc. (DMTF): Cloud Infrastructure Management Interface - Common Information Model (CIMI-CIM) A CIM Representation of the CIMI Model. URL https://www.dmtf.org/sites/default/files/standards/documents/DSP0264_1.0.0.pdf. - abgerufen am 2018-08-07

[Dist14] Distributed Management Task Force, Inc.: Common Information Model (CIM) Metamodel 3.0.1. URL https://www.dmtf.org/sites/default/files/standards/documents/DSP0004_3.0.1.pdf. - abgerufen am 2016-09-02

[Dist16] Distributed Management Task Force, Inc.: Common Information Model. URL http://www.dmtf.org/standards/cim. - abgerufen am 2016-02-09

[DuNa09] Dumitraş, T.; Narasimhan, P.: Why Do Upgrades Fail and What Can We Do About It?: Toward Dependable, Online Upgrades in Enterprise System. In: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware, Middleware ’09. New York, NY, USA: Springer-Verlag New York, Inc., 2009, S. 18:1–18:20

[DwAC98] Dwyer, M. B.; Avrunin, G. S.; Corbett, J. C.: Property Specification Patterns for Finite-state Verification. In: Proceedings of the Second Workshop on Formal Methods in Software Practice, FMSP ’98. New York, NY, USA: ACM, 1998, S. 7–15

[EcBr09] Eckert, M.; Bry, F.: Complex Event Processing (CEP). In: Informatik-Spektrum Bd. 32 (2009), Nr. 2, S. 163–167

[EHPB02] Ehrig, H.; Hoffmann, K.; Padberg, J.; Baldan, P.; Heckel, R.: High-Level Net Processes. In: Brauer, W.; Ehrig, H.; Karhumäki, J.; Salomaa, A. (Hrsg.): Formal and Natural Computing, Lecture Notes in Computer Science: Springer Berlin Heidelberg, 2002, S. 191–219

[Elli14] Elliot, S.: DevOps and the cost of downtime: Fortune 1000 best practice metrics quantified. In: International Data Corporation (IDC) (2014)

[ElNu93] Ellis, C. A.; Nutt, Gary J.: Modeling and enactment of workflow systems. In: Application and Theory of Petri Nets 1993, Lecture Notes in Computer Science: Springer, Berlin, Heidelberg, 1993, S. 1–16

[ETHP14] Elgammal, A.; Turetken, O.; Heuvel van den, W.-J.; Papazoglou, Mike: Formalizing and appling compliance patterns for business process compliance. In: Software & Systems Modeling (2014), S. 1–28

[FeFB13] Feitelson, D. G.; Frachtenberg, E.; Beck, K. L.: Development and deployment at Facebook. In: IEEE Internet Computing (2013), Nr. 4, S. 8–17

[Fers79] Ferstl, O.K.: Konstruktion und Analyse von Simulationsmodellen. Beiträge zur Datenverarbeitung und Unternehmensforschung. Bd. 22: Hain, Königstein/Ts., 1979

[FeSi08] Ferstl, O.K.; Sinz, E.J.: Grundlagen der Wirtschaftsinformatik. 6. Aufl.: München, 2008

[FHKF09] Frank, U.; Heise, D.; Kattenstroth, H.; Ferguson, D.; Hadar, E.; Waschke, M.: ITML: a domain-specific modeling language for supporting business driven it management. In: Proceedings of the 9th Workshop on Domain-Specific Modeling (DSM) at the International Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA), Orlando, Florida, USA, 2009

[FKLS17] Fellmann, M.; Koschmider, A.; Laue, R.; Schoknecht, A.; Vetter, A.: A Taxonomy and Catalog of Business Process Model Patterns. In: Proceedings of the 22Nd European Conference on Pattern Languages of Programs, EuroPLoP ’17. New York, NY, USA: ACM, 2017, S. 19:1–19:10

[FKLS18] Fellmann, M.; Koschmider, A.; Laue, R.; Schoknecht, A.; Vetter, A.: Business process model patterns: state-of-the-art, research classification and taxonomy. In: Business Process Management Journal (2018)

[FSWG15] Farshchi, M.; Schneider, J.-G.; Weber, I.; Grundy, J.: Experience Report: Anomaly Detection of Cloud Application Operations Using Log and Cloud Metric Correlation Analysis. In: 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), 2015, S. 24–34

[FSWG17] Farshchi, M.; Schneider, J.-G.; Weber, I.; Grundy, J.: Metric selection and anomaly detection for cloud operations using log and metric correlation analysis. In: Journal of Systems and Software (2017)

[GaDa10] Galup, S. D.; Dattero, R.: A five-step method to tune your ITSM processes. In: Information Systems Management Bd. 27 (2010), Nr. 2, S. 156–167

[GDQC09] Galup, S. D.; Dattero, R.; Quan, J. J.; Conger, S.: An Overview of IT Service Management. In: Commun. ACM Bd. 52 (2009), Nr. 5, S. 124–127

[GeLa81] Genrich, H. J.; Lautenbach, K.: System modelling with high-level Petri nets. In: Theoretical Computer Science, Special Issue Semantics of Concurrent Computation. Bd. 13 (1981), Nr. 1, S. 109–135

[Genr86] Genrich, H. J.: Predicate/Transition Nets. In: Brauer, W.; Reisig, W.; Rozenberg, G. (Hrsg.): Petri Nets: Central Models and Their Properties, Lecture Notes in Computer Science: Springer Berlin Heidelberg, 1986, S. 207–247

[GHJV94] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design patterns: elements of reusable object-oriented software: Pearson Education, 1994

[GHKR01] Garschhammer, M.; Hauck, R.; Kempter, B.; Radisic, I.; Roelle, H.; Schmidt, H.: The MNM service model—refined views on generic service management. In: Communications and Networks, Journal of Bd. 3 (2001), Nr. 4, S. 297–306

[GHLP14] Gunawi, H. S.; Hao, M.; Leesatapornwongsa, T.; Patana-anake, T.; Do, T.; Adityatama, J.; Eliazar, K. J.; Laksono, A.; u. a.: What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems. In: Proceedings of the ACM Symposium on Cloud Computing, SOCC '14. New York, NY, USA: ACM, 2014

[Göss05] Gössinger, R.: Dienstleistungen als Problemlösungen: Eine produktionstheoretische Analyse auf der Grundlage von Eigenschaften, Information - Organisation - Produktion: Deutscher Universitätsverlag, 2005

[Gray86] Gray, J.: Why do computers stop and what can be done about it? In: Symposium on reliability in distributed software and database systems: Los Angeles, CA, USA, 1986, S. 3–12

[GŽBP14] Gama, J.; Žliobaité, I.; Bifet, A.; Pechenizkiy, M.; Bouchachia, A.: A Survey on Concept Drift Adaptation. In: ACM Comput. Surv. Bd. 46 (2014), Nr. 4, S. 44:1–44:37

[HaKo13] Halang, W. A.; Konakovsky, R. M.: Sicherheitsgerichtete Echtzeitsysteme. 2. Aufl.: Springer Vieweg, 2013

[HBSZ15] Huang, P.; Bolosky, W. J.; Singh, A.; Zhou, Y.: ConfValley: a systematic configuration validation framework for cloud services. In: Proceedings of the Tenth European Conference on Computer Systems: ACM, 2015, S. 19

[HeGS15] Hernantes, J.; Gallardo, G.; Serrano, N.: IT Infrastructure-Monitoring Tools. In: IEEE Software Bd. 32 (2015), Nr. 4, S. 88–93

[HoSt09] Hoyer, V.; Stanoevska-Slabeva, K.: Generic Business Model Types for Enterprise Mashup Intermediaries. In: Nelson, M. L.; Shaw, M. J.; Strader, T. J. (Hrsg.): Value Creation in E-Business Management, Lecture Notes in Business Information Processing: Springer Berlin Heidelberg, 2009, S. 1–17

[HoZB04] Hochstein, A.; Zarnekow, R.; Brenner, W.: ITIL als Common-Practice-Referenzmodell für das IT-Service-Management—Formale Beurteilung und Implikationen für die Praxis. In: Wirtschaftsinformatik Bd. 46 (2004), Nr. 5, S. 382–389

[Inte06] International Organization for Standardization: International Standard - ISO/IEC 14764 IEEE Std 14764-2006 Software Engineering #2013; Software Life Cycle Processes #2013; Maintenance. In: ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998) (2006), S. 1–46

[KBBL12] Kopp, O.; Binz, T.; Breitenbücher, U.; Leymann, F.: BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications. In: Mendling, J.; Weidlich, M. (Hrsg.): Business Process Model and Notation, Lecture Notes in Business Information Processing: Springer Berlin Heidelberg, 2012, S. 38–52

[KeKS01] Keller, A.; Kreger, H.; Schopmeyer, K.: Towards a CIM Schema for RunTime Application Management. In: Operations & Management: 12th International Workshop on Distributed Systems. Nancy, France: Institut National de Recherche en Informatique et en Automatique, INRIA, 2001

[Kell05] Keller, A.: Automating the change management process with electronic contracts. In: Seventh IEEE International Conference on E-Commerce Technology Workshops, 2005, S. 99–107

[KeNS92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“, Institut für Wirtschaftsinformatik (IWi) im Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI GmbH), Universität des Saarlandes, Saarbrücken - Publikationen: Institut für Wirtschaftsinformatik, 1992

[KHWW04] Keller, A.; Hellerstein, J. L; Wolf, J. L; Wu, K-L; Krishnan, V.: The CHAMPS system: change management with planning and scheduling. In: IEEE/IFIP Network Operations and Management Symposium. Bd. 1, 2004, S. 395-408 Vol.1

[KiWa04] Kiciman, E.; Wang, Y.-M.: Discovering correctness constraints for self-management of system configuration. In: International Conference on Autonomic Computing, 2004. Proceedings., 2004, S. 28–35

[Klei98] Kleinaltenkamp, M.: Begriffsabgrenzungen und Erscheinungsformen von Dienstleistungen. In: Handbuch Dienstleistungsmanagement: Gabler Verlag, Wiesbaden, 1998, S. 29–52

[Kona88] Konakovsky, R.: Verfahren der Vollständigen Fehlererkennung Durch Gezielten Einsatz von Diversität. In: Lauber, R. (Hrsg.): Prozeßrechensysteme ’88, Informatik-Fachberichte: Springer Berlin Heidelberg, 1988, S. 281–290

[Kosc07] Koschmider, A.: Ähnlichkeitsbasierte Modellierungsunterstützung für Geschäftsprozesse: KIT Scientific Publishing, 2007

[KoTP09] Koschmider, A.; Torres, V.; Pelechano, V.: Elucidating the mashup hype: Definition, challenges, methodical guide and tools for mashups. In: 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web in conjunction with the 18th International World Wide Web Conference. Madrid, 2009, S. 1–9

[Krcm15] Krcmar, H.: Informationsmanagement. 6. Aufl.: Springer Berlin Heidelberg, 2015

[Lapr95] Laprie, J.-C.: Dependable computing and fault-tolerance : Concepts and Terminology. In: Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, ’ Highlights from Twenty-Five Years’., 1995, S. 2–11

[LaWR14] Lanz, A.; Weber, B.; Reichert, M.: Time patterns for process-aware information systems. In: Requirements Engineering Bd. 19 (2014), Nr. 2, S. 113–141

[LeBe02] Lee, J. K.; Ben-Natan, R.: Integrating Service Level Agreements: Optimizing Your OSS for SLA Delivery: John Wiley & Sons, 2002

[LeFo14] Lewis, J.; Fowler, M.: Microservices. URL https://martinfowler.com/articles/microservices.html. - abgerufen am 2018-11-01

[Leim12] Leimeister, J. M.: Dienstleistungsengineering und -management. Berlin Heidelberg: Springer-Verlag, 2012

[Lenz03] Lenz, K.: Modellierung und Ausführung von E-Business-Prozessen mit XML-Netzen: VWF Verlag für Wissenschaft und Forschung GmbH, 2003

[LeOb02] Lenz, K.; Oberweis, Andreas: Integrierte Dokumenten-und Ablaufmodellierung von E-Business-Prozessen. In: Promise, 2002, S. 40–51

[LeOb03] Lenz, K.; Oberweis, A.: Inter-organizational Business Process Management with XML Nets. In: Petri Net Technology for Communication-Based Systems, Lecture Notes in Computer Science: Springer, Berlin, Heidelberg, 2003, S. 243–263

[LMMR13] Ly, L. T.; Maggi, F. M.; Montali, M.S.anie; Aalst van der, W. M. P.: A Framework for the Systematic Comparison and Evaluation of Compliance Monitoring Approaches. In: 2013 17th IEEE International Enterprise Distributed Object Computing Conference: IEEE, 2013, S. 7–16

[LSAB11] Luckham, D.; Schulte, W. R.; Adkins, J.; Bizarro, P.; Jacobsen, H.-A.; Mavashev, A.; Michelson, B. M.; Niblett, P.; u. a.: Event Processing Glossary – Version 2.0 (2011)

[Luck01] Luckham, D. C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001

[Lude02] Ludewig, J.: Modelle Im Software Engineering - Eine Einführung Und Kritik. In: Modellierung in Der Praxis - Modellierung Für Die Praxis, Modellierung 2002: GI, 2002, S. 7–22

[LwKO15] Lwakatare, L. E.; Kuvaja, P.; Oivo, M.: Dimensions of DevOps. In: Lassenius, C.; Dingsøyr, T.; Paasivaara, M. (Hrsg.): Agile Processes, in Software Engineering, and Extreme Programming, Lecture Notes in Business Information Processing: Springer International Publishing, 2015, S. 212–217

[MaHG15] Maes, K.; Haes De, S.; Grembergen Van, W.: Developing a Value Management Capability: A Literature Study and Exploratory Case Study. In: Information Systems Management Bd. 32 (2015), Nr. 2, S. 82–104

[Male13] Maleri, R.: Grundlagen der Dienstleistungsproduktion: Springer-Verlag, 2013

[MeBr06] Meffert, H.; Bruhn, M.: Dienstleistungsmarketing: Grundlagen - Konzepte - Methoden.Mit Fallstudien. 5. Aufl.: Gabler Verlag, 2006

[MeNA07] Mendling, J.; Neumann, G.; Aalst van der, W. M. P.: Understanding the Occurrence of Errors in Process Models Based on Metrics. In: On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, Lecture Notes in Computer Science: Springer, Berlin, Heidelberg, 2007, S. 113–130

[Mevi06] Mevius, M.: Kennzahlenbasiertes Management von Geschäftsprozessen mit Petri-Netzen. München: Dr. Hut, 2006

[MGDA10] Madduri, V. R.; Gupta, M.; De, P.; Anand, V.: Towards Mitigating Human Errors in IT Change Management Process. In: Maglio, P. P.; Weske, M.; Yang, J.; Fantinato, M. (Hrsg.): Service-Oriented Computing, Lecture Notes in Computer Science: Springer Berlin Heidelberg, 2010, S. 657–662

[Mode11] Model, Business Process: Notation (BPMN). URL https://www.omg.org/spec/BPMN/2.0. - abgerufen am 2016-06-24. — OMG Specification. Object Management Group

[NOBM04] Nagaraja, K.; Oliveira, F.; Bianchini, R.; Martin, R. P.; Nguyen, T. D.: Understanding and Dealing with Operator Mistakes in Internet Services. In: Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6, OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, S. 61–76

[Oasi13a] OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. URL http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html. - abgerufen am 2016-01-19

[Oasi13b] OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0. URL http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.pdf. - abgerufen am 2016-06-09

[Oasi13c] OASIS: Topology and Orchestration Specification for Cloud Applications Version 1.0 XML Schema. URL http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/schemas/TOSCA-v1.0.xsd. - abgerufen am 2016-09-14

[Oasi17] OASIS: Instance Model for TOSCA Version 1.0. URL http://docs.oasis-open.org/tosca/TOSCA-Instance-Model/v1.0/TOSCA-Instance-Model-v1.0.html. - abgerufen am 2018-07-02

[Ober96] Oberweis, A.: Modellierung und Ausführung von Workflows mit Petri-Netzen. Wiesbaden: Vieweg+Teubner Verlag, 1996

[Offi11a] Office, Stationery: ITIL Service Transition 2011. London: The Stationery Office Ltd, 2011

[Offi11b] Office, Stationery: ITIL Service Strategy 2011. 2. Aufl. London: The Stationery Office Ltd, 2011

[Offi11c] Office, The Cabinet: ITIL Service Design 2011. 2. Aufl. London: The Stationery Office Ltd, 2011

[Offi11d] Office, Stationery: ITIL Service Operation 2011. 2. Aufl. Norwich: The Stationery Office Ltd, 2011

[Offi11e] Office, Stationery: ITIL Continual Service Improvement 2011. 2. Aufl. Norwich: The Stationery Office Ltd, 2011

[ONBB06] Oliveira, F.; Nagaraja, K.; Bachwani, R.; Bianchini, R.; Martin, R. P.; Nguyen, T. D.: Understanding and Validating Database System Administration. In: USENIX Annual Technical Conference, General Track: Boston, MA, 2006, S. 213–228

[Open16] OpenStack: Heat Orchestration Template (HOT) specification. URL https://docs.openstack.org/heat/rocky/template_guide/hot_spec.html. - abgerufen am 2016-08-31

[OpGP03] Oppenheimer, D.; Ganapathi, A.; Patterson, D. A.: Why Do Internet Services Fail, and What Can Be Done About It? In: Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4, USITS'03. Berkeley, CA, USA: USENIX Association, 2003

[PeNa05] Pertet, S.; Narasimhan, Priya: Causes of failure in web applications. In: Parallel Data Laboratory (2005)

[Petr62] Petri, C. A.: Kommunikation mit Automaten. Mathematisches Institut der Universität Bonn, Technische Hochschule Darmstadt, 1962

[PLSW08] Palatin, N.; Leizarowitz, A.; Schuster, A.; Wolff, R.: Mining for misconfigured machines in grid systems. In: Data Mining Techniques in Grid Computing Environments (2008), S. 71

[Powe11] Powers, D. M. W.: Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. In: Journal of Machine Learning Technologies Bd. 2 (2011), Nr. 1, S. 37–63

[RAHE05] Russell, N.; Aalst van der, W. M. P.; Hofstede ter, A. H. M.; Edmond, D.: Workflow Resource Patterns: Identification, Representation and Tool Support. In: Advanced Information Systems Engineering, Lecture Notes in Computer Science: Springer, Berlin, Heidelberg, 2005, S. 216–232

[RaGr12] Rance, S.; Great Britain: Cabinet Office: Key Element Guide Itil Service Transition: Aligned to the 2011 Editions 2nd edition: TSO, 2012

[Reis13] Reisig, W.: Petrinetze: Eine Einführung: Springer-Verlag, 2013

[RHEA05] Russell, N.; Hofstede ter, A: H. M.; Edmond, D.; Aalst van der, W. M. P.: Workflow Data Patterns: Identification, Representation and Tool Support. In: Delcambre, L. M. L.; Kop, C.; Mayr, H. C.; Mylopoulos, J.; Pastor, O. (Hrsg.): 24th International Conference on Conceptual Modeling, Lecture Notes in Computer Science. Bd. 3716. Klagenfurt, Austria: Springer, 2005, S. 353–368

[RiZü96] Riehle, D.; Züllighoven, H.: Understanding and Using Patterns in Software Development. In: Theor. Pract. Object Syst. Bd. 2 (1996), Nr. 1, S. 3–13

[Scot99] Scott, D.: Making smart investments to reduce unplanned downtime. In: Tactical Guidelines Research Note Note TG-07-4033, Gartner Group, Stamford, CT (1999)

[SVFO16] Schoknecht, A.; Vetter, A:; Fill, H.-G.; Oberweis, A.: Using the Horus Method for Succeeding in Business Process Engineering Projects. In: Domain-Specific Conceptual Modeling: Springer iv, Cham, 2016, S. 127–147

[ŚwBo17] Święcicki, B.; Borzemski, L.: How Is Server Software Configured? Examining the Structure of Configuration Files. In: Information Systems Architecture and Technology: Proceedings of 38th International Conference on Information Systems Architecture and Technology – ISAT 2017, Advances in Intelligent Systems and Computing: Springer, Cham, 2017, S. 217–229

[Thom16] Thomas, S.: thomasspatzier/tosca-tools. URL ttps://github.com/thomasspatzier/tosca-tools. - abgerufen am 2016-09-07

[TrFL09] Trastour, D.; Fink, R.; Liu, F.: Changerefinery: Assisted refinement of high-level IT change requests. In: IEEE International Symposium on Policies for Distributed Systems and Networks: IEEE, 2009, S. 68–75

[Valm98] Valmari, A.: The state explosion problem. In: Reisig, W.; Rozenberg, G. (Hrsg.): Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science: Springer Berlin Heidelberg, 1998, S. 429–528

[Vett15] Vetter, A.: Vermeidung von IT-Störungen unter Einsatz eines Kontrollsystems zur Durchführung von IT-Changes. In: INFORMATIK 2015. Bonn: Gesellschaft für Informatik e.V., 2015, S. 1795–1800

[Vett16] Vetter, A.: Detecting Operator Errors in Cloud Maintenance Operations. In: 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2016, S. 639–644

[Vett17] Vetter, A.: Detecting Operator Errors In Cloud Computing Using Anti-Patterns. In: Proceedings of the 7th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2017). Neuchatel: ceur-ws.org, 2017

[Vett19] Vetter, A.: Online Detection Of Operator Errors In Cloud Computing Using Anti-Patterns. In: Ceravolo, P.; Guetl, C.; Rinderle-Ma, S. (Hrsg.): Data-Driven Process Discovery and Analysis, Lecture Notes in Business Information Processing: Springer International Publishing, 2019

[Vinc16] Vincent, P.: CEP Tooling Market Survey 2016. URL http://www.complexevents.com/2016/05/12/cep-tooling-market-survey-2016/. - abgerufen am 2018-04-03

[WBLA09] Wickboldt, J. A.; Bianchin, L. A.; Lunardi, R. C.; Andreis, F. Girardi; Cordeiro, W. L. D. C.; Both, C. B.; Granville, L. Z.; Gaspary, L. P.; u. a.: Improving IT Change Management Processes with Automated Risk Assessment. In: Integrated Management of Systems, Services, Processes and People in IT: Springer, 2009, S. 71–84

[WBLG11] Wickboldt, J. A.; Bianchin, L. A.; Lunardi, R. C.; Granville, L. Z.; Gaspary, L. P.; Bartolini, C.: A framework for risk assessment based on analysis of historical information of workflow execution in IT systems. In: Computer Networks Bd. 55 (2011), Nr. 13, S. 2954–2975

[Wc14] W3C: XQuery 3.0: An XML Query Language. URL https://www.w3.org/TR/xquery-30/. - abgerufen am 2016-08-29

[WeBL14] Wettinger, J.; Breitenbücher, U.; Leymann, F.: Standards-Based DevOps Automation and Integration Using TOSCA. In: Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC '14. Washington, DC, USA: IEEE Computer Society, 2014, S. 59–68

[WeMW09] Weidlich, M.; Mendling, J.; Weske, M.: Computation of behavioural profiles of process models. In: Technischer Bericht, Hasso Plattner Institute for IT-Systems Engineering (2009)

[WiCE09] Winniford, M.; Conger, S.; Erickson-Harris, L.: Confusion in the ranks: IT service management practice and terminology. In: Information Systems Management Bd. 26 (2009), Nr. 2, S. 153–163

[WMCL09] Wickboldt, J. A.; Machado, G. S.; Cordeiro, W. L. D. C.; Lunardi, R. C.; Santos, A. D. dos; Andreis, F. G.; Both, C. B.; Granville, L. Z.; u. a.: A solution to support risk analysis on IT Change Management. In: 2009 IFIP/IEEE International Symposium on Integrated Network Management, 2009, S. 445–452

[Wso17] WSO2: WSO2 Complex Event Processor Documentation Version 4.2.0. URL https://docs.wso2.com/display/CEP420/SiddhiQL+Guide+3.1. - abgerufen am 2017-10-24

[WZMG11] Weidlich, M.; Ziekow, H.; Mendling, J.; Günther, O.; Weske, M.; Desai, N.: Event-based monitoring of process execution violations. In: Business Process Management: Springer, 2011, S. 182–198

[XuZh15] Xu, T.; Zhou, Y.: Systems Approaches to Tackling Configuration Errors: A Survey. In: ACM Comput. Surv. Bd. 47 (2015), Nr. 4, S. 70:1–70:41

[XZWB14] Xu, X.; Zhu, L.; Weber, I.; Bass, L.; u. a.: POD-diagnosis: Error diagnosis of sporadic operations on cloud applications. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks: IEEE, 2014, S. 252–263

[YMZZ11] Yin, Z.; Ma, X.; Zheng, J.; Zhou, Y.; Bairavasundaram, L. N.; Pasupathy, S.: An Empirical Study on Configuration Errors in Commercial and Open Source Systems. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP '11. New York, NY, USA: ACM, 2011, S. 159–172

[YXPY11] Yuan, D.; Xie, Y.; Panigrahy, R.; Yang, J.; Verbowski, C.; Kumar, A.: Context-based online configuration-error detection. In: Proceedings of the 2011 USENIX conference on USENIX annual technical conference: USENIX Association, 2011, S. 28–28

[ZaHB06] Zarnekow, R.; Hochstein, A.; Brenner, W.: Serviceorientiertes IT-Management: ITIL-Best-Practices Und -Fallstudien (Business Engineering). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006

[ZhLi11] Zhang, G.; Liu, L.: Why do migrations fail and what can we do about it? In: Proceedings of the 25th USENIX Large Installation System Administration Conference (LISA'11), 2011

Anhang: Anti-Pattern als SiddhiQL-Ereignisregeln

```
1  from eingehende_VorgangEreignisse [(app == VORGAENGER.appakt and
   op == VORGAENGER.opakt and prop == VORGAENGER.propakt and
   value == VORGAENGER.valueakt) in VORGAENGER]
2  insert into #temp;
3  from #temp as tmp join VORGAENGER as vor
4  on tmp.app == vor.appakt and tmp.op == vor.opakt and tmp.prop == vor.propakt
   and tmp.value == vor.valueakt
5  select tmp.app, tmp.op, tmp.prop, tmp.value, vor.appvor, vor.opvor, vor.propvor,
   vor.valuevor, vor.endevor
6  insert into #temp1;
7  from #temp1 [not((appvor == VorgangEreignisHistorie.app and
   opvor == VorgangEreignisHistorie.op and propvor == VorgangEreignisHistorie.prop
   and valuevor == VorgangEreignisHistorie.value) in VorgangEreignisHistorie)]
8  select str:concat("Bevor die Aktivitaet ",app, ", ", op, ", ", prop, ", ", value, " ausge-
   fuehrt werden kann, muss die Aktivitaet ", appvor, " ,", opvor, ", ", propvor, ", ",
   valuevor, " durchgefuehrt worden sein.") as Fehlermeldung
9  insert into Fehlermeldungen_Stream;
```

Quelltext 9.1: VORGÄNGER als SiddhiQL-Ereignisregel

```
1  from eingehende_VorgangEreignisse [(app == DirektePaare.appnachf and
   op == DirektePaare.opnachf and prop == DirektePaare.propnachf and
   value == DirektePaare.valuenachf) in DirektePaare]
2  insert into #temp;
3  from #temp#window.length(1) as tmp join VorgangEreignisHistorie as veh
4  on time:timestampInMilliseconds(veh.timestamp, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") < time:timestampInMilliseconds(tmp.timestamp, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'") and tmp.st == veh.st
5  select tmp.app, tmp.op, tmp.prop, tmp.value, max(time:timestampInMillisec-
   onds(veh.timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")) as timestampvor, veh.app
   as appvor, veh.op as opvor, veh.prop as propvor, veh.value as valuevor
6  insert current events into #temp1;
7  from #temp1 [not ((app == DirektePaare.appnachf and op == DirektePaare.opnachf
   and prop == DirektePaare.propnachf and value == DirektePaare.valuenachf and
   appvor == DirektePaare.appvor and opvor == DirektePaare.opvor and
   propvor == DirektePaare.propvor and valuevor == DirektePaare.valuevor) in
   DirektePaare)]
8  select app, op, prop, value
9  insert into #temp2;
10 from #temp2 as tmp join DirektePaare as vor
11 on tmp.app == vor.appnachf and tmp.op == vor.opnachf and
   tmp.prop == vor.propnachf and tmp.value == vor.valuenachf
12 select str:concat("Die notwendige Vorgaenger-Aktivitaet ",vor.appvor, ", ",
   vor.opvor, ", ", vor.propvor, ", ", vor.valuevor, " wurde nicht direkt vor der Aktivi-
   taet ", tmp.app, ", ",tmp.op, ", ", tmp.prop, ", ", tmp.value, " durchgefuehrt.") as
   Fehlermeldung
13 insert into Fehlermeldungen_Stream;
```

Quelltext 9.2: DIREKTER VORGÄNGER als SddhiQL-Ereignisregel

```
1  from Zeittrigger join ANWESENHEIT
2  select app, op, prop, start, value, ende, time:timestampInMilliseconds() >
   time:timestampInMilliseconds(ende, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") as
   zeitpunkt
3  insert into #temp;
4  from #temp [zeitpunkt == true]
5  insert into #temp1;
6  from #temp1#window.length(1) as tmp join ANWESENHEIT as anw
7  on tmp.app == anw.app and tmp.op == anw.op and tmp.prop == anw.prop and
   tmp.value == anw.value and time:timestampInMilliseconds(tmp.start, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") <= time:timestampInMilliseconds(anw.start, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") and time:timestampInMilliseconds(tmp.ende, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") >= time:timestampInMilliseconds(anw.ende, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'")
8  select tmp.app, tmp.op, tmp.prop, tmp.value, tmp.start, tmp.ende, count
   (tmp.app, tmp.op, tmp.prop, tmp.value) as sollanzahl
9  insert into temp_stream_pattern1;
10 from temp_stream_pattern1#window.firstUnique(app, op, prop, value, start, ende,
   sollanzahl) as tmp left outer join VorgangEreignisHistorie as veh
11 on tmp.app == veh.app and tmp.op == veh.op and tmp.prop == veh.prop and
   tmp.value == veh.value and time:timestampInMilliseconds(veh.timestamp, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'") >= time:timestampInMilliseconds(tmp.start, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'") and time:timestampInMilliseconds(veh.timestamp,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <= time:timestampInMilliseconds(tmp.ende,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")
12 select tmp.app, tmp.op, tmp.prop, tmp.value, ifThenElse(veh.timestamp IS NULL,
   convert(0,'long'),count(veh.timestamp)) as istanzahl, sollanzahl, tmp.start,
   tmp.ende
13 group by tmp.app, tmp.op, tmp.prop, tmp.value, tmp.start, tmp.ende
14 insert into #temp2;
15 from #temp2 [sollanzahl > istanzahl]
16 select str:concat("Die Aktivitaet ", app,", ", op, ", ",prop, ", ",value, " wurde zwi-
   schen ", start, " und ", ende, " nur ", istanzahl, " mal, statt ", sollanzahl, " mal durch-
   gefuehrt.") as Fehlermeldung
17 insert into Fehlermeldungen_Stream;
```

Quelltext 9.3: ANWESENHEIT als SiddhiQL-Ereignisregel

```
1  from Zeittrigger join ALTERNATIVE_ANWESENHEIT
2  select appakt, opakt, propakt, valueakt, startakt, endeakt, time:timestampInMilli-
   seconds() >  time:timestampInMilliseconds(endeakt, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") as zeitpunkt
3  insert into #temp;
4  from #temp [zeitpunkt == true]
5  insert into #temp1;
6  from #temp1#window.length(1) as tmp join ALTERNATIVE_ANWESENHEIT as anw
7  on tmp.appakt == anw.appakt and tmp.opakt == anw.opakt and
   tmp.propakt == anw.propakt and tmp.valueakt == anw.valueakt and
   time:timestampInMilliseconds(tmp.startakt, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <=
   time:timestampInMilliseconds(anw.startakt, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") and
   time:timestampInMilliseconds(tmp.endeakt, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <=
   time:timestampInMilliseconds(anw.endeakt, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")
8  select tmp.appakt, tmp.opakt, tmp.propakt, tmp.valueakt, tmp.startakt, tmp.en-
   deakt, distinctcount (anw.startakt) as sollanzahl
9  insert into #temp2;
10 from #temp2#window.firstUnique(appakt, opakt, propakt, valueakt, startakt,
   endeakt, sollanzahl) as tmp left outer join VorgangEreignisHistorie as veh
11 on tmp.appakt == veh.app and tmp.opakt == veh.op and tmp.propakt == veh.prop
   and tmp.valueakt == veh.value and time:timestampInMilliseconds(veh.timestamp,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") >= time:timestampInMilliseconds(tmp.startakt,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") and time:timestampInMillisec-
   onds(veh.timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <= time:timestampInMilli-
   seconds(tmp.endeakt, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")
12 select tmp.appakt,tmp.opakt, tmp.propakt, tmp.valueakt, ifThen-
   Else(veh.timestamp IS NULL, convert(0,'long'),count(veh.timestamp)) as istanzahl,
   sollanzahl, tmp.startakt, tmp.endeakt
13 group by tmp.appakt, tmp.opakt, tmp.propakt, tmp.valueakt, tmp.startakt,
   tmp.endeakt, sollanzahl
14 insert into #temp3;
15 from #temp3 [sollanzahl > istanzahl]
16 insert into #temp4;
17 from #temp4 as tmp join ALTERNATIVE_ANWESENHEIT as alt
```

```
18 on tmp.appakt == alt.appakt and tmp.opakt == alt.opakt and
   tmp.propakt == alt.propakt and tmp.valueakt == alt.valueakt and
   tmp.startakt == alt.startakt and tmp.endeakt == alt.endeakt
19 select alt.appakt, alt.opakt, alt.propakt, alt.valueakt, alt.startakt, alt.endeakt,
   alt.appalt, alt.opalt, alt.propalt, alt.valuealt, alt.startalt, alt.endealt, istanzahl,
   sollanzahl
20 insert into #temp5;
21 from #temp5 as tmp join ALTERNATIVE_ANWESENHEIT as alt
22 on tmp.appalt == alt.appakt and tmp.opalt == alt.opakt and
   tmp.propalt == alt.propakt and tmp.valuealt == alt.valueakt and
   tmp.startalt == alt.startakt and tmp.endealt == alt.endeakt
23 select tmp.appakt as app, tmp.opakt as op, tmp.propakt as prop, tmp.valueakt as
   value, tmp.startakt as start, tmp.endeakt as ende, alt.appakt, alt.opakt, alt.propakt,
   alt.valueakt, alt.startakt, alt.endeakt, alt.appalt, alt.opalt, alt.propalt, alt.valuealt,
   alt.startalt, alt.endealt, istanzahl, sollanzahl
24 insert into #temp6;
25 from #temp6 as tmp left outer join VorgangEreignisHistorie as veh
26 on (tmp.appakt == veh.app and tmp.opakt == veh.op and tmp.propakt == veh.prop
   and tmp.valueakt == veh.value and time:timestampInMilliseconds(tmp.startakt,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <= time:timestampInMillisec-
   onds(veh.timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") and time:timestampInMil-
   liseconds(tmp.endeakt, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") >=
   time:timestampInMilliseconds(veh.timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"))
   or (tmp.appalt == veh.app and tmp.opalt == veh.op and tmp.propalt == veh.prop
   and tmp.valuealt == veh.value and time:timestampInMilliseconds(tmp.startalt,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <= time:timestampInMillisec-
   onds(veh.timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") and time:timestampInMil-
   liseconds(tmp.endealt, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") >= time:timestampInMil-
   liseconds(veh.timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"))
27 select ifThenElse(veh.timestamp IS NULL, time:timestampInMilliseconds('2100-12-
   31T23:59:59.000000Z', "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"), time:timestampInMilli-
   seconds(veh.timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")) as auftreten, tmp.app,
   tmp.op, tmp.prop, tmp.value, tmp.start, tmp.ende, istanzahl, sollanzahl
28 insert into #temp7;
29 from #temp7#window.time(1 sec)
30 select min(auftreten) as minauftreten, app, op, prop,value, start, ende, istanzahl,
   sollanzahl
```

```
31 group by app, op, prop, value, start, ende, istanzahl, sollanzahl
32 insert expired events into #temp8;
33 from #temp8#window.firstUnique(app, op, prop, value, start, ende, istanzahl,
   sollanzahl)
34 insert current events into #temp9;
35 from #temp9[minauftreten == time:timestampInMilliseconds('2100-12-
   31T23:59:59.000000Z', "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")]
36 select str:concat("Die Aktivitaet ", app,", ", op, ", ",prop, ", ",value, " wurde zwi-
   schen ", start, " und ", ende, " nur ", istanzahl, " mal, statt ", sollanzahl, " mal durch-
   gefuehrt.") as Fehlermeldung
37 insert current events into ausgehende_Fehlermeldungen;
38 from #temp9[minauftreten != time:timestampInMilliseconds('2100-12-
   31T23:59:59.000000Z', "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")]
39 insert into #temp10;
40 from #temp10 as tmp join VorgangEreignisHistorie as veh
41 on tmp.minauftreten == time:timestampInMilliseconds(veh.timestamp, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'")
42 select tmp.app, tmp.op, tmp.prop, tmp.value, tmp.start, tmp.ende, veh.app as ap-
   palt, veh.op as opalt, veh.prop as propalt, veh.value as valuealt, tmp.istanzahl,
   tmp.sollanzahl
43 insert into #temp11;
44 from #temp11 [not((app == ALTERNATIVE_ANWESENHEIT.appakt and
   op == ALTERNATIVE_ANWESENHEIT.opakt and prop == ALTERNATIVE_ANWESEN-
   HEIT.propakt and value == ALTERNATIVE_ANWESENHEIT.valueakt and
   start == ALTERNATIVE_ANWESENHEIT.startakt and ende == ALTERNATIVE_ANWE-
   SENHEIT.endeakt and appalt == ALTERNATIVE_ANWESENHEIT.appalt and
   opalt == ALTERNATIVE_ANWESENHEIT.opalt and propalt == ALTERNATIVE_ANWE-
   SENHEIT.propalt and valuealt == ALTERNATIVE_ANWESENHEIT.valuealt) in ALTER-
   NATIVE_ANWESENHEIT)]
45 select str:concat("Die Aktivitaet ", app,", ", op, ", ",prop, ", ",value, " wurde zwi-
   schen ", start, " und ", ende, " nur ", istanzahl, " mal, statt ", sollanzahl, " mal durch-
   gefuehrt.") as Fehlermeldung
46 insert into Fehlermeldungen_Stream;
```

Quelltext 9.4: ALTERNATIVE ANWESENHEIT als SiddhiQL-Ereignisregel

```
1  from eingehende_VorgangEreignisse [(app == ABWESENHEIT.app and
   op == ABWESENHEIT.op and prop == ABWESENHEIT.prop and
   value == ABWESENHEIT.value and time:timestampInMilliseconds(timestamp,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") >= time:timestampInMilliseconds(ABWESEN-
   HEIT.start, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") and time:timestampInMillisec-
   onds(timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <= time:timestampInMillisec-
   onds(ABWESENHEIT.ende, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")) in ABWESENHEIT]
2  insert into #temp;
3  from #temp#window.length(1) as tmp join ABWESENHEIT as abw
4  on tmp.app == abw.app and tmp.op == abw.op and tmp.prop == abw.prop and
   tmp.value == abw.value and time:timestampInMilliseconds(abw.start, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") <= time:timestampInMilliseconds(tmp.timestamp, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'") and time:timestampInMilliseconds(tmp.timestamp,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <= time:timestampInMilliseconds(abw.ende,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")
5  select tmp.app, tmp.op, tmp.prop,tmp.value, abw.start, abw.ende, count(abw.app,
   abw.op, abw.prop, abw.value) as sollanzahl
6  insert current events into #temp1;
7  from #temp1#window.length(1) as tmp join VorgangEreignisHistorie as veh
8  on tmp.app == veh.app and tmp.op == veh.op and tmp.prop == veh.prop and
   tmp.value == veh.value and time:timestampInMilliseconds(veh.timestamp, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'") >= time:timestampInMilliseconds(tmp.start, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'") and time:timestampInMilliseconds(veh.timestamp,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <= time:timestampInMilliseconds(tmp.ende,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")
9  select tmp.app, tmp.op, tmp.prop, tmp.value, tmp.start, tmp.ende,
   count(veh.timestamp) as istanzahl, sollanzahl
10 insert current events into #temp2;
11 from #temp2 [istanzahl > sollanzahl]
12 select str:concat("Die Aktivitaet ",app, ", ",op, ", ", prop, ", ", value, " wurde zwi-
   schen ",start, " und ", ende, " " ,istanzahl, " mal statt maximal ",sollanzahl, " mal
   durchgefuehrt.") as Fehlermeldung
13 insert into Fehlermeldungen_Stream;
```

Quelltext 9.5: ABWESENHEIT Bedingung 1 als SiddhiQL-Ereignisregel

```
1  from Vorgang_Ereignisse_Stream [((app == ABWESENHEIT.app and
   op == ABWESENHEIT.op and prop == ABWESENHEIT.prop and
   value == ABWESENHEIT.value) in ABWESENHEIT)]
2  insert into #temp;
3  from #temp [not((app == ABWESENHEIT.app and op == ABWESENHEIT.op and prop
   == ABWESENHEIT.prop and value == ABWESENHEIT.value and
   time:timestampInMilliseconds(timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") >
   time:timestampInMilliseconds(ABWESENHEIT.start, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") and time:timestampInMilliseconds(timestamp, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") < time:timestampInMilliseconds(ABWESENHEIT.ende,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")) in ABWESENHEIT)]
4  select str:concat("Die Aktivität ",app, ", ",op, ", ", prop, ", ", value, " wurde außer-
   halb eines geplanten Zeitraums durchgeführt.") as Fehlermeldung
5  insert into Fehlermeldungen_Stream;
```

Quelltext 9.6: ABWESENHEIT Bedingung 2 als SiddhiQL-Ereignisregel

```
1  from Vorgang_Ereignisse_Stream [not((app == ABWESENHEIT.app and
   op == ABWESENHEIT.op and prop == ABWESENHEIT.prop) in ABWESENHEIT)]
2  select str:concat("Die Aktivität ", app, ", ", op, ", ", prop, " sollte nicht durchgeführt
   werden.") as Fehlermeldung
3  insert into Fehlermeldungen_Stream;
```

Quelltext 9.7: ABWESENHEIT Bedingung 3 als SiddhiQL-Ereignisregel

```
1  from eingehende_VorgangEreignisse [(app == ALTERNATIVE_ANWESENHEIT.ap-
   pakt and op == ALTERNATIVE_ANWESENHEIT.opakt and prop == ALTERNA-
   TIVE_ANWESENHEIT.propakt and value == ALTERNATIVE_ANWESENHEIT.val-
   ueakt and time:timestampInMilliseconds(timestamp, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") >= time:timestampInMilliseconds(ALTERNATIVE_AN-
   WESENHEIT.startakt, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") and
   time:timestampInMilliseconds(timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <=
   time:timestampInMilliseconds(ALTERNATIVE_ANWESENHEIT.endeakt, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'")) in ALTERNATIVE_ANWESENHEIT]
2  insert into #temp;
```

```
3  from #temp as tmp join ALTERNATIVE_ANWESENHEIT as alt
4  on tmp.app == alt.appakt and tmp.op == alt.opakt and tmp.prop == alt.propakt
   and tmp.value == alt.valueakt and time:timestampInMillisec-
   onds(tmp.timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") >=
   time:timestampInMilliseconds(alt.startakt, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")
   and time:timestampInMilliseconds(tmp.timestamp, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") <= time:timestampInMilliseconds(alt.endeakt, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'")
5  select alt.appakt, alt.opakt, alt.propakt, alt.valueakt, alt.startakt, alt.endeakt,
   alt.appalt, alt.opalt, alt.propalt, alt.valuealt, alt.startalt, alt.endealt,
   tmp.timestamp
6  insert into #temp1;
7  from #temp1 as tmp join ALTERNATIVE_ANWESENHEIT as alt
8  on tmp.appalt == alt.appakt and tmp.opalt == alt.opakt and
   tmp.propalt == alt.propakt and tmp.valuealt == alt.valueakt and
   tmp.startalt == alt.startakt and tmp.endealt == alt.endeakt
9  select tmp.appakt as app, tmp.opakt as op, tmp.propakt as prop, tmp.valueakt
   as value, tmp.startakt as start, tmp.endeakt as ende, tmp.timestamp, alt.appakt,
   alt.opakt, alt.propakt, alt.valueakt, alt.startakt, alt.endeakt, alt.appalt, alt.opalt,
   alt.propalt, alt.valuealt, alt.startalt, alt.endealt
10 insert into #temp2;
11 from #temp2 as tmp join VorgangEreignisHistorie as veh
12 on (tmp.appakt == veh.app and tmp.opakt == veh.op and
   tmp.propakt == veh.prop and tmp.valueakt == veh.value and
   time:timestampInMilliseconds(tmp.startakt, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")
   <= time:timestampInMilliseconds(veh.timestamp, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") and time:timestampInMilliseconds(tmp.endeakt, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'") >= time:timestampInMilliseconds(veh.timestamp,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")) or (tmp.appalt == veh.app and tmp.opalt ==
   veh.op and tmp.propalt == veh.prop and tmp.valuealt == veh.value and
   time:timestampInMilliseconds(tmp.startalt, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")
   <= time:timestampInMilliseconds(veh.timestamp, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") and time:timestampInMilliseconds(tmp.endealt, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'") >= time:timestampInMilliseconds(veh.timestamp,
   "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"))
```

```
13 select time:timestampInMilliseconds(veh.timestamp, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") as auftreten, tmp.app, tmp.op, tmp.prop, tmp.value,
   tmp.timestamp
14 insert into #temp3;
15 from #temp3#window.time(1 sec)
16 select min(auftreten) as minauftreten, app, op, prop, value, timestamp
17 group by app, op, prop, value, timestamp
18 insert current events into #temp4;
19 from #temp4#window.firstUnique(app, op, prop, value, timestamp)
20 insert current events into #temp5;
21 from #temp5 as tmp join VorgangEreignisHistorie as veh
22 on tmp.minauftreten == time:timestampInMilliseconds(veh.timestamp, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'")
23 select tmp.app, tmp.op, tmp.prop,tmp.value, tmp.timestamp, veh.app as ap-
   palt, veh.op as opalt, veh.prop as propalt, veh.value as valuealt
24 insert into #temp6;
25 from #temp6 [(app == ALTERNATIVE_ANWESENHEIT.appakt and
   op == ALTERNATIVE_ANWESENHEIT.opakt and prop == ALTERNATIVE_AN-
   WESENHEIT.propakt and time:timestampInMilliseconds(timestamp, "yyyy-MM-
   dd'T'HH:mm:ss.SSS'Z'") >= time:timestampInMilliseconds(ALTERNATIVE_AN-
   WESENHEIT.startakt, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") and
   time:timestampInMilliseconds(timestamp, "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'") <=
   time:timestampInMilliseconds(ALTERNATIVE_ANWESENHEIT.endeakt, "yyyy-
   MM-dd'T'HH:mm:ss.SSS'Z'") and appalt == ALTERNATIVE_ANWESENHEIT.appalt
   and opalt == ALTERNATIVE_ANWESENHEIT.opalt and propalt == ALTERNA-
   TIVE_ANWESENHEIT.propalt) in ALTERNATIVE_ANWESENHEIT]
26 select str:concat("Die Aktivitaet ", app,", ", op, ", ",prop, ", ",value, " darf nicht
   ausgefuehrt werden, da bereits die Aktivitaet ", appalt, ", ", opalt, ", ", propalt, ",
   ", valuealt, " durchgefuehrt wurde.") as Fehlermeldung
27 insert into Fehlermeldungen_Stream;
```

Quelltext 9.8: ALTERNATIVE ABWESENHEIT als SiddhiQL-Ereignisregel

```
1  from Vorgang_Ereignisse_Stream [(WERT.app == app and WERT.op == op and
   WERT.prop == prop) in WERT]
2  insert into #temp;
3  from #temp [not((WERT.app == app and WERT.op == op and WERT.prop == prop
   and WERT.value == value) IN WERT)]
4  insert into #temp1;
5  from #temp1 as tmp join WERT as wer
6  on tmp.app == wer.app and tmp.op == wer.op and tmp.prop == wer.prop and
   tmp.value != wer.value
7  select str:concat("Der geänderte Wert ", tmp.value, " entspricht nicht dem geplan-
   ten Wert ", wer.value, ".") as Fehlermeldung
8  insert into Fehlermeldungen_Stream;
```

Quelltext 9.9: WERT als SiddhiQL-Ereignisregel

```
1  from eingehende_VorgangEreignisse [(app == BEDINGUNGEN.app and
   op == BEDINGUNGEN.op and prop == BEDINGUNGEN.prop and
   value == BEDINGUNGEN.value) in BEDINGUNGEN]
2  insert into #temp;
3  from #temp as tmp join BEDINGUNGEN as bed
4  on tmp.app == bed.app and tmp.op == bed.op and tmp.prop == bed.prop and
   tmp.value == bed.value
5  select tmp.app, tmp.op, tmp.prop, tmp.value, bed.zapp, bed.state
6  insert into #temp1;
7  from #temp1 [not((zapp == ZustandEreignisHistorie.app and
   state == ZustandEreignisHistorie.state) in ZustandEreignisHistorie)]
8  select str:concat("Fuer die Durchfuehrung der Aktivitaet ", app, ", ",op, ", ", prop, ",
   ", value, " muss zunaechst die folgende IT-Service-Komponenten in den jeweiligen
   Status versetzt werden: ", zapp, ", ",state, ".") as Fehlermeldung
9  insert into Fehlermeldungen_Stream;
```

Quelltext 9.10: ZUSTAND als SiddhiQL-Ereignisregel

Wartungsfehler sind einer der häufigsten Gründe für IT-Service-Ausfälle und können bei IT-Service-Anbietern Schäden in Millionenhöhe verursachen. Es kann angenommen werden, dass durch anhaltende Trends wie Cloud Computing und DevOps die Anzahl an IT-Service-Wartungen weiter zunehmen wird und somit auch potenziell die Anzahl an Wartungsfehlern. In diesem Buch wird eine Methode vorgestellt, die es IT-Service-Anbietern ermöglicht, Wartungsfehler automatisiert während der Durchführung einer IT-Service-Wartung zur Laufzeit zu erkennen. Dadurch können Wartungsfehler behoben werden, bevor sie Schaden anrichten können.

ISBN 978-3-7315-0921-9

Gedruckt auf FSC-zertifiziertem Papier