

Universität Stuttgart

Eine musterbasierte Methode zur Automatisierung des Anwendungsmanagements

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der
Universität Stuttgart zur Erlangung der Würde eines Doktors der
Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von
Uwe Breitenbücher
aus Ludwigsburg

Hauptberichter: Prof. Dr. Dr. h. c. Frank Leymann

Mitberichter: Univ. Prof. Dr. Schahram Dustdar
Prof. Dr. Florian Daniel

Tag der mündlichen Prüfung: 29. März 2016

Institut für Architektur von Anwendungssystemen
der Universität Stuttgart

2016

INHALTSVERZEICHNIS

1	Einleitung & Motivation	11
1.1	Vision der Arbeit	14
1.2	Problemstellung und Forschungsbeiträge	16
1.3	Veröffentlichungen im Rahmen der Arbeit	22
1.4	Aufbau der Arbeit	24
2	Grundlagen des Anwendungsmanagements und verwandte Arbeiten	25
2.1	Modelle zur Beschreibung von Anwendungen	26
2.2	Anwendungsmanagement	38
2.3	Managementparadigmen und -technologien	48
2.4	Muster in der Informationstechnologie	90
2.5	Zusammenfassung	97
3	Musterbasierte Methode zum Management von Anwendungen	99
3.1	Die Rolle des Anwendungsmanagers	100
3.2	Anforderungen an eine Managementmethode	101
3.3	PALMA-Methode	103
3.4	Variante zur Kombination mehrerer Managementmuster	112
3.5	Anwendung und Automatisierung der PALMA-Methode	113
3.6	Diskussion und Umsetzung der erhobenen Anforderungen	117

4	Deklarative Sprache zur Modellierung von Managementaufgaben	119
4.1	Research Design der Entwicklung von DMMN	121
4.2	DMMN-Metamodell	122
4.3	Vordefinierte Relationen, Annotationen und Schnittstellen	152
4.4	Grafische Notation	155
4.5	Serialisierung von DMMN-Modellen	162
4.6	Diskussion	163
5	Generierung von Managementplänen	165
5.1	Idee und Grundkonzept	166
5.2	Management-Planlets	168
5.3	Generische Management-Planlets	176
5.4	Orchestrierung von Management-Planlets	181
5.5	Transformation in ein imperatives Managementmodell	225
5.6	Diskussion und geplante Forschungsarbeiten	232
6	Automatisierte Anwendung von Managementmustern	239
6.1	Idee und Grundkonzept	240
6.2	Automatisierte Managementmuster	242
6.3	Anwendung automatisierter Managementmuster	245
6.4	Beispiel eines automatisierten Managementmusters	253
6.5	Unterschiedliche Abstraktionshöhen automatisierter Muster	256
6.6	Managementprimitive	259
6.7	Automatisierte Muster zur Bereitstellung von Anwendungen	260
6.8	Diskussion und Abgrenzung zu verwandten Arbeiten	266
7	Kontextsensitive Analyse deklarativer Managementaufgaben	271
7.1	Motivation	272
7.2	Idee und Grundkonzept	274
7.3	Automatisierte Analysemuster	276
7.4	Diskussion und Abgrenzung zu verwandten Arbeiten	284

8	Architektur und Prototyp des PALMA-Frameworks	287
8.1	Architektur des PALMA-Frameworks	288
8.2	Prototypische Implementierung des PALMA-Frameworks	293
8.3	Verwendung des PALMA-Frameworks in anderen Arbeiten	296
9	Zusammenfassung und Ausblick	297
	Danksagungen	300
	Literaturverzeichnis	301
	Abbildungsverzeichnis	337
	Symbolverzeichnis	340

ZUSAMMENFASSUNG

Das Management laufender Geschäftsanwendungen gehört zu den kritischen Vorgängen des IT-Betriebs, da Unachtsamkeiten zu fehlerhaften Anwendungszuständen und zum Ausfall ganzer Anwendungslandschaften führen können. Speziell die manuelle Managementdurchführung birgt aufgrund zunehmend unüberschaubarer Anwendungsstrukturen, unbekannter Abhängigkeiten zwischen Komponenten, unzureichender Dokumentation sowie komplexer Managementwerkzeuge ein stetig größer werdendes wirtschaftliches Risiko. Aus diesem Grund wurden in den vergangenen Jahren neue Paradigmen und eine große Anzahl an Technologien zur Automatisierung des Anwendungsmanagements entwickelt. Die zunehmende Komplexität von Architekturen durch verteilte Systeme, Virtualisierung von Komponenten, Cloud Computing und dem aufkommenden Internet der Dinge erfordert jedoch immer häufiger die Kombination mehrerer dieser Managementtechnologien, um übergeordnete Managementziele bezüglich eines komplexen IT-Systems umzusetzen. Dabei treten sowohl (i) konzeptionelle als auch (ii) technische Fragestellungen auf, die von unterschiedlichen Expertengruppen in Kollaboration analysiert und gelöst werden müssen. Die Integration dieser beiden Abstraktionsebenen stellt dabei eine grundlegende Herausforderung dar, die im gegenwärtigen Anwendungsmanagement aufgrund fehlender durchgängiger Automatisierungsmöglichkeiten zumeist unter hohem Aufwand individuell angenommen werden muss.

Um diese Automatisierungslücke zwischen den beiden Abstraktionsebenen zu schließen, stellt diese Arbeit eine hybride, musterbasierte Managementmethode namens PALMA vor. Die Methode kombiniert das deklarative mit dem imperativen Managementparadigma und ermöglicht dadurch die automatisierte Anwendung generischer Managementmuster. Häufig auftretende Managementprobleme, wie beispielsweise die Migration einer Anwendungskomponente in eine Cloud-Umgebung unter Wahrung deren Verfügbarkeit, können mittels automatisierten Managementmustern effizient für individuelle Anwendungen gelöst und die zugehörigen Prozesse automatisiert ausgeführt werden. Die Methode unterstützt die Kollaboration von Experten und kann manuell, semi-automatisiert sowie vollständig automatisiert angewendet werden.

Zur Umsetzung der Methode wird eine deklarative Sprache namens DMMN vorgestellt, welche die Modellierung von Managementaufgaben auf hoher deklarativer Abstraktionsebene unter Ausblendung technischer Ausführungsdetails ermöglicht. Bei der automatisierten Ausführung von Managementmustern werden deklarative Managementmodelle in dieser Sprache generiert, welche die jeweilige Musterlösung für die betroffene Anwendung in Form der auszuführenden Managementaufgaben abstrakt spezifizieren. Zu deren Ausführung wird ein Transformationsverfahren vorgestellt, das deklarative Managementmodelle in ausführbare, imperative Prozessmodelle übersetzt. Die Generierung dieser Prozessmodelle basiert dabei auf der Orchestrierung wiederverwendbarer Managementbausteine, die in Form von Subprozessen modelliert sind und als Management-Planlets bezeichnet werden. Durch diese Transformation werden die Stärken beider Paradigmen vereint und umfangreiches Management komplexer IT-Systeme ermöglicht. Zudem wird ein musterbasiertes Verfahren vorgestellt, mithilfe dessen Managementaufgaben in deklarativen Managementmodellen automatisch auf Probleme analysiert und diesbezüglich korrigiert werden können. Dadurch wird die Korrektheit der Durchführung gewährleistet und der Systemadministrator bei der Modellierung der Modelle unterstützt.

Die in dieser Arbeit vorgestellten Konzepte werden im Rahmen des sogenannten PALMA-Frameworks prototypisch implementiert, um die praktische Umsetzbarkeit der theoretischen Konzepte und Ansätze zu validieren.

ABSTRACT

Managing running business applications belongs to the critical issues of IT operation as mistakes and carelessness quickly lead to faulty application states, system downtime, and failures of whole application landscapes. Especially the manual execution of management tasks poses a steadily increasing risk due to complex application structures, unknown dependencies, inadequate documentation, and non-trivial management tools. For this reason, different management paradigms and a broad variety of management automation technologies have emerged in the past years. However, the increasing complexity of application architectures of distributed systems, the broad virtualization introduced by Cloud Computing, as well as the upcoming trend of the Internet of Things (IoT) often requires a combination of multiple heterogeneous management technologies to execute complex holistic management processes. Unfortunately, this combination is not well-tackled by existing approaches due to missing end-to-end automation and, therefore, requires cooperating experts to solve a variety of issues on the (i) conceptual as well as on the (ii) technical layer. Especially integrating these two layers for creating individual application-specific solutions typically requires huge manual effort.

This dissertation presents a hybrid pattern-based management method called PALMA to bridge this gap. The method combines the declarative and the imperative management paradigm for automating the application of management patterns to solve frequently occurring management problems efficiently for individual applications, for example, to migrate an application component to a Cloud environment without downtime. The PALMA method is technology-independent, supports the collaboration of different experts, and can be executed manually, semi-automatically, as well as fully automatically.

To apply the PALMA method, this dissertation presents a declarative modeling language called DMMN for describing management tasks in a declarative and abstract manner without the need to deal with low-level technical execution details. For automating the application of management patterns, a concept is introduced to automatically generate DMMN models that declaratively describe a pattern's solution in the form of abstract management tasks to be executed for individual applications. To automatically detect and correct modeling faults in DMMN models as well as to optimize declaratively specified management tasks, a pattern-based analysis and adaptation concept is presented. This concept supports the creation and refinement of correct DMMN models. As DMMN models specify management tasks only declaratively, a transformation method is introduced to automatically generate executable imperative management process models. The generation of these process models is based on orchestrating reusable management building blocks called Management-Planlets, which provide small sub-processes that execute a formally described set of declarative management tasks on a predefined application structure. Due to this model transformation, the strengths of both paradigms can be combined, which provides the basis for automating the management of complex IT systems.

To validate the practical feasibility of the concepts presented in this dissertation, a prototype called PALMA-Framework was developed.

EINLEITUNG & MOTIVATION

Der erfolgreiche Einsatz von Informationstechnologie (IT) spielt sowohl für Unternehmen als auch für Privatpersonen eine immer größer werdende Rolle. Während die Produktivität vieler Unternehmen direkt von der effizienten Ausführung IT-gestützter Geschäftsprozesse abhängig ist, beeinflussen Softwaresysteme immer stärker das alltägliche Leben - teils indirekt, wenn Systeme wie die elektronische Anzeigetafel verspäteter Züge in unsere Lebensumgebung integriert werden, zum Teil wird diese Einflussnahme jedoch zunehmend direkter, wenn beispielsweise Cloud-Dienste und mobile Überwachungssysteme für die Analyse und Sicherung unserer körperlichen Gesundheitswerte eingesetzt werden. Aufgrund dieser Beobachtungen kann die Hypothese aufgestellt werden, dass die Menschheit zunehmend von der erfolgreichen Ausführung allgegenwärtiger Informationstechnologie abhängig wird. Ohne zu bewerten, ob dies eine realistische Einschätzung darstellt oder ob man diese Entwicklung befürwortet oder nicht, gewinnt die Verfügbarkeit und die korrekte Funktionsweise eingesetzter Informationstechnologie unbestritten für viele Menschen und Organisationen eine stetig zunehmende Bedeutung. Das Management und die Wartung von Informationstechnologie gehören daher zu den kritischen

Abläufen des Anwendungsbetriebs, da kleinste Fehler oder Unachtsamkeiten zu fehlerhaften Funktionen oder, im schlimmsten Fall, zum Ausfall einer gesamten Anwendungslandschaft führen können. Speziell die manuelle Managementdurchführung ist bei unüberschaubaren Anwendungsstrukturen, unbekannten Abhängigkeiten zwischen Komponenten, unzureichender Dokumentation sowie komplexen Managementwerkzeugen eine riskante Aufgabe. Neben technischen Herausforderungen, wie der Bedienung von Management-APIs, der Implementierung und Ausführung von Skripten oder der Orchestrierung verschiedener Technologien, stellen konzeptionelle Probleme die zweite große Ursache der Managementkomplexität dar. Fragen, wie beispielsweise eine Anwendungskomponente ohne Ausfallzeit in eine Cloud-Umgebung migriert werden kann, sind häufig nicht trivial zu beantworten und erfordern tiefes Expertenwissen und große Erfahrung im Bereich der jeweiligen Managementdomäne.

Zur Lösung dieser zum einen (i) *technischen Herausforderungen* und zum anderen (ii) *konzeptionellen Probleme* wurden in zahlreichen wissenschaftlichen und industriellen Projekten, Publikationen und Kollaborationen Ansätze erarbeitet. Daraus entstanden eine Vielzahl an Managementtechnologien, welche die Ausführung von Managementaufgaben auf technischer Ebene automatisieren und damit zur Reduktion manueller Fehler beitragen. Beispiele dafür sind Werkzeuge für das Konfigurationsmanagement, die beispielsweise durch Chef [Ops16] und Puppet [Pup16] weitverbreitete Technologien darstellen und im gegenwärtigen IT-Management große Beachtung erfahren. Neben diesen, typischerweise Open-Source, Werkzeugen entwickelte die Industrie zahlreiche Produkte und Dienste, welche vor allem für größere Unternehmen von fundamentaler Bedeutung sind, da meist deren gesamte Anwendungsmanagementprozesse von diesen Werkzeugen abhängig sind oder auf diesen aufbauen. Neben technischen Managementlösungen wurde in der angewandten Wissenschaft große Aufmerksamkeit auf die konzeptionelle Erfassung und Formalisierung existierender Managementexpertise gelegt. Dafür wurden vor allem im Bereich der Mustersprachen (engl. „*pattern languages*“) grundlegende Arbeiten publiziert, die für häufig wiederkehrende Probleme anwendbares Expertenwissen auf hoher Abstraktionsebene in Form bewährter Lösungen dokumentieren. Ne-

ben bekannten, etablierten Mustersprachen, die seit Jahren zur effizienten Anwendungsentwicklung [GHJV94; BMR+96; Fow02] und Anwendungsintegration [HW04] eingesetzt werden, wird zunehmend das Management von Informationstechnologie unterstützt [FLR+13; FLR+14; JPCL14].

Eine grundsätzliche Lücke zwischen der konzeptionellen und der technischen Abstraktionsebene erschwert jedoch signifikant den praktischen Einsatz von Mustersprachen im Bereich des Anwendungsmanagements: Um ein Managementmuster zur Lösung eines bestimmten Problems einzusetzen, muss dessen abstrakt beschriebene Lösung für den jeweiligen Anwendungsfall verfeinert und auf konkrete Technologien, Schnittstellen und domänenspezifische Sprachen (DSLs) abgebildet werden [BBKL14c]. In der Praxis wird dies manuell durchgeführt, wozu enorme Expertise und Erfahrung bezüglich der involvierten Managementtechnologien erforderlich sind, da menschliche Fehler schnell zum Ausfall ganzer Anwendungslandschaften führen können [BP01; Gad14; Fei15]. Speziell die Kombination mehrerer heterogener Managementtechnologien, die häufig in einer Vielzahl zur Umsetzung der abstrakt beschriebenen Lösungen kombiniert werden müssen, stellt dabei eine riskante und wissensintensive Herausforderung dar [BBK+13a; LTC+14]. Soll beispielsweise eine Anwendungs-komponente von einer existierenden Infrastruktur in eine Cloud-Umgebung migriert werden, müssen dazu typischerweise die Managementdienste des Cloud-Anbieters mit Konfigurationsmanagementtechnologien wie Chef oder Puppet kombiniert werden. Aufgrund unterschiedlicher Managementparadigmen, Schnittstellen und Datenformaten resultiert daraus jedoch eine enorme Komplexität [BBK+13a; WBB+14a]. Die manuelle Anwendung eines Managementmusters ist daher zeitaufwändig, fehleranfällig und somit nicht effizient umsetzbar [BBKL14c]. Des Weiteren wird das Wissen, welches Anwender bei der Musteranwendung erlangen, selten dokumentiert und muss folglich bei einer erneuten Umsetzung desselben Managementmusters von Grund auf neu erlernt werden. Der Mustereinsatz zur Lösung von Managementproblemen ist daher mit komplexen Herausforderungen verbunden, die sowohl bei der Umsetzung eines Managementmusters als auch bei der Dokumentation wie diese Umsetzung durchgeführt wurde zu großen Effizienzproblemen führen.

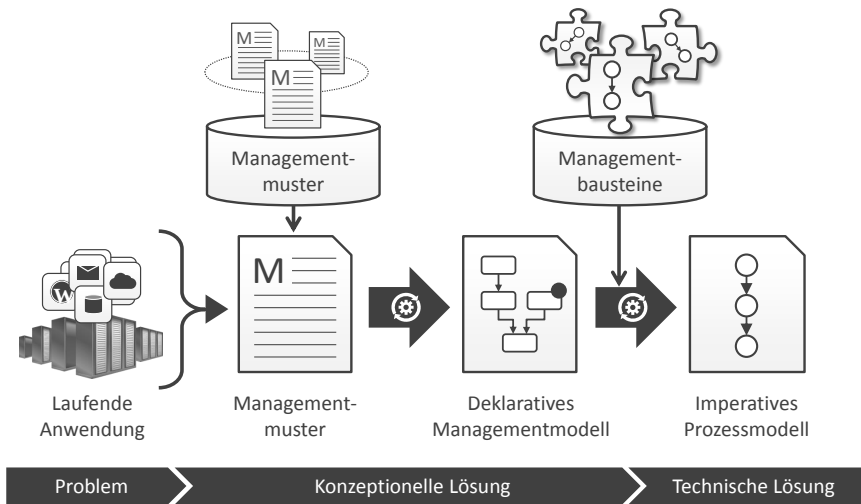


Abbildung 1.1: Vision der musterbasierten Managementautomatisierung

1.1 Vision der Arbeit

Die Vision der vorliegenden Arbeit löst diese Probleme durch eine hybride, musterbasierte Methode, die konzeptionelle Lösungen in Form deklarativer Managementmodelle generiert und diese durch Orchestrierung wiederverwendbarer Managementbausteine in ausführbare imperative Prozessmodelle transformiert. [Abbildung 1.1](#) illustriert diese Vision: Ausgehend von einem bestimmten Managementproblem, beispielsweise wie eine laufende Anwendung ohne Ausfallzeit in eine Cloud-Umgebung migriert werden kann, wird ein geeignetes Managementmuster ausgewählt, das eine bewährte Lösung für dieses Problem auf konzeptioneller Ebene dokumentiert. Auf Basis dessen abstrakter Musterlösung wird ein deklaratives Managementmodell generiert, welches die auszuführenden Managementaufgaben auf identischer Abstraktionshöhe für die betroffene Anwendung beschreibt. Das generierte deklarative Modell beinhaltet ausschließlich die Spezifikation *was* erreicht werden soll, jedoch ohne die konkreten Ausführungsdetails zu beschreiben, *wie* die deklarierten Aufgaben technisch umzusetzen sind. Daher ist dieses Modell nicht direkt ausführbar

und wird im folgenden Schritt durch Orchestrierung von Managementbausteinen, welche als Subprozesse modelliert sind, in ein ausführbares imperatives Prozessmodell transformiert. Das resultierende Modell beschreibt alle auszuführenden technischen Aktivitäten, um das Managementproblem für die betroffene Anwendung zu lösen, und kann daher vollautomatisiert ausgeführt werden.

Die Vision ermöglicht dadurch das automatisierte Lösen wiederkehrender Managementprobleme durch den Einsatz von Mustern und integriert die Anwendung konzeptionellen Wissens mit dessen automatisierter technischer Umsetzung. Dabei kann konzeptionelle Managementexpertise strukturiert als Muster formalisiert und dadurch wiederverwendbar für zukünftige Managementaufgaben erfasst werden. Gleichmaßen ermöglicht die Wiederverwendung ausführbarer Subprozesse, welche in dieser Vision als Managementbausteine dienen, die stetige Optimierung der Gesamtprozesse durch lokale, von Experten durchgeführte, technische Verbesserungen. Durch die Integration beider Abstraktionsebenen kann Managementwissen somit sowohl auf (i) konzeptioneller als auch auf (ii) technischer Ebene automatisiert werden, wodurch die Effizienz des Anwendungsmanagements signifikant verbessert, Fehlerquellen reduziert und Expertenkooperationen ermöglicht werden. Unternehmen können durch dieses Konzept eigene Managementexpertise unabhängig von individuellen Anwendungen erfassen und automatisieren sowie externes Wissen gezielt in automatisierter Form von Dritten einkaufen. Die Transformation in Prozessmodelle ermöglicht eine robuste und verlässliche Managementdurchführung, wodurch Fehlerquellen der manuellen Ausführung sowie unzuverlässiger Managementsysteme reduziert werden. Die Ebene der deklarativen Beschreibung ermöglicht zudem die abstrakte Modellierung von Managementaufgaben unabhängig der letztendlich eingesetzten Managementtechnologien, wodurch die Komplexität und Fehleranfälligkeit signifikant reduziert werden im Vergleich zu existierenden Ansätzen, welche Managementaufgaben typischerweise auf feingranularer, technischer Ebene beschreiben. Trotz dieser hohen Abstraktion können Administratoren die technische Ebene in ihrer Gesamtheit beeinflussen, da die deklarative Beschreibung in Prozessmodelle transformiert werden, welche beliebig vor deren Ausführung adaptiert und konfiguriert werden können.

1.2 Problemstellung und Forschungsbeiträge

Basierend auf dieser Vision werden in diesem Abschnitt die verschiedenen Problemstellungen des Anwendungsmanagements im Detail beschrieben und es wird vorgestellt, wie die sechs Forschungsbeiträge der vorliegenden Arbeit diese Problemstellungen adressieren. In der folgenden Abbildung wird ein Überblick der Forschungsbeiträge gegeben: Forschungsbeitrag 1 stellt eine musterbasierte Managementmethode namens PALMA vor, welche durch die Beiträge 2-5 umgesetzt und automatisiert werden kann. Forschungsbeitrag 6 umfasst die Architektur und prototypische Implementierung des PALMA-Frameworks, welches die PALMA-Methode und die anderen Forschungsbeiträge realisiert.

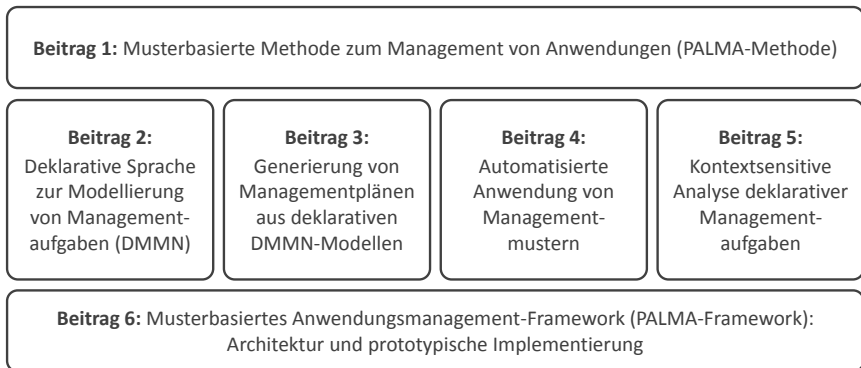


Abbildung 1.2: Überblick der Forschungsbeiträge

Die Forschungsbeiträge 2, 3 und 5 können dabei auch unabhängig von der vorgestellten Managementmethode eingesetzt werden, um (i) Aufgaben des Anwendungsmanagements deklarativ zu modellieren, (ii) Managementaufgaben kontextsensitiv in ihrem Gesamtkontext zu analysieren und (iii) deklarative Managementmodelle in ausführbare imperative Prozessmodelle zu übersetzen. Das entwickelte Management-Framework unterstützt dazu jeden der Beiträge separat. In den folgenden Abschnitten werden die einzelnen Forschungsbeiträge sowie deren Zusammenhänge und Abhängigkeiten im Detail vorgestellt.

1.2.1 Musterbasierte Methode zum Management von Anwendungen

Zahlreiche Vorgehensmodelle zum Management von Anwendungen basieren auf konzeptionell beschriebenen Lösungen, welche die technische Ausführung den Anwendern überlassen, oder technischen Ansätzen, welche Lösungen auf rein technischer Implementierungsebene umsetzen und die Verantwortung der Konzeption dem Anwender übertragen. Während konzeptionelle Ansätze, wie beispielsweise das Musterkonzept, Lösungen unabhängig von bestimmten Technologien dokumentieren, sind viele der technischen Ansätze proprietäre Lösungen und somit an bestimmte Technologien, Infrastrukturen oder Software gekoppelt. Ein Vorgehensmodell, das konzeptionelle Managementlösungen auf Basis von Mustern unabhängig von Technologien mit deren automatisierten Ausführung für individuelle Anwendungsfälle unterstützt, ist gegenwärtig nicht vorhanden. In dieser Arbeit wird daher eine modellgetriebene Methode vorgestellt, die beide Ansätze auf Basis von Managementmustern integriert.

Forschungsbeitrag 1 (Musterbasierte Methode zum Management von Anwendungen). Die vorliegende Arbeit beschreibt eine hybride Managementmethode namens PALMA (*Pattern-based Application Management*), die durch Kombination des deklarativen Managementansatzes mit dem imperativen Managementansatz die Anwendung von Mustern zum Management laufender Anwendung ermöglicht. Dazu werden deklarative Managementmodelle eingeführt, welche die abstrakte Deklaration auszuführender Managementaufgaben in Abhängigkeit ihres Kontextes ermöglichen. Die Methode ist (i) vollständig technologieunabhängig, (ii) unterstützt die Kollaboration unterschiedlicher Expertenrollen und kann (iii) sowohl manuell, semi-automatisiert sowie vollständig automatisiert durch die weiteren Forschungsbeiträge der vorliegenden Arbeit umgesetzt werden. Dabei kann die Managementausführung sowohl technisch als auch konzeptionell zu jedem Zeitpunkt beeinflusst werden, sodass Anwender, z. B. Systemadministratoren, jederzeit überwachen und manuell eingreifen können.

1.2.2 Deklarative Sprache zur Modellierung von Managementaufgaben

Zur Automatisierung des Anwendungsmanagements existieren verschiedene Paradigmen und Technologien, die grundsätzlich in zwei Klassen eingeteilt werden können: (i) Deklarative Ansätze und (ii) imperative Ansätze. Deklarative Ansätze ermöglichen die Modellierung von Managementaufgaben durch Spezifikation der zu erreichenden Ziele, fern deren technischer Umsetzung. Im Gegensatz dazu beschreiben imperative Ansätze, wie beispielsweise Skripte oder Workflow-Modelle, alle technischen Details, die zur automatisierten Ausführung der gewünschten Managementaufgaben benötigt werden, d. h. den Kontroll- und Datenfluss zwischen den auszuführenden Aktivitäten. Während existierende Arbeiten die beiden Modellierungsstile meist getrennt voneinander betrachten, kombiniert die vorliegende Arbeit beide Paradigmen zur automatisierten Umsetzung des ersten Forschungsbeitrags, wozu eine geeignete Sprache für die deklarative Modellierung von Managementaufgaben erforderlich ist.

Forschungsbeitrag 2 (Deklarative Sprache zur Modellierung von Managementaufgaben). Im zweiten Forschungsbeitrag wird eine deklarative Sprache namens DMMN (*Declarative Application Management Modelling and Notation*) vorgestellt. Das DMMN-Metamodell ermöglicht die deklarative Modellierung zustandsmodifizierender sowie zustandserhaltender Managementaufgaben in Abhängigkeit der Struktur und dem aktuellen Zustand der betroffenen Anwendung. Managementaufgaben können mittels DMMN abstrakt und unabhängig der Technologien, die letztendlich zu deren Ausführung eingesetzt werden, modelliert werden. Die Sprache definiert zudem eine grafische Notation, welche die visuelle Modellierung deklarativer Managementaufgaben auf Grundlage graphbasierter Topologiemodelle ermöglicht. DMMN kann grundsätzlich zur manuellen Modellierung von Managementaufgaben verwendet werden, der Fokus dieser Arbeit liegt jedoch auf der Verwendung zur Automatisierung der PALMA-Methode.

1.2.3 Generierung von Managementplänen aus deklarativen DMMN-Modellen

Deklarative Managementbeschreibungen spezifizieren Managementziele, die für eine bestimmte Anwendung erreicht werden sollen. Sie beschreiben jedoch ausschließlich *was* erreicht werden soll, ohne Informationen zu enthalten, *wie* die gesetzten Ziele erreicht bzw. technisch umgesetzt werden können. Daher sind deklarative DMMN-Modelle aufgrund fehlender Informationen nicht direkt ausführbar und müssen entweder (i) interpretiert werden, um die Ausführungslogik abzuleiten, oder (ii) in ausführbare imperative Prozessmodelle übersetzt werden, welche sowohl den Kontrollfluss, den Datenfluss sowie alle technischen Anweisungen spezifizieren, die zur Umsetzung der deklarativen Managementbeschreibung ausgeführt werden müssen.

Forschungsbeitrag 3 (Generierung von Managementplänen aus deklarativen DMMN-Modellen). Im dritten Forschungsbeitrag wird ein Verfahren vorgestellt, das deklarative DMMN-Modelle vollautomatisiert in ausführbare imperative Prozessmodelle transformiert. Die generierten Prozessmodelle werden in der vorliegenden Arbeit als *Managementpläne* bezeichnet und mit Hilfe der Workflow-Technologie realisiert, wodurch eine robuste und verlässliche Managementdurchführung gewährleistet wird. Zur Generierung dieser Pläne werden wiederverwendbare Workflow-Modelle namens *Management-Planlets* eingeführt, die eine abgeschlossene Menge atomarer Managementaufgaben ausführen und aufgrund der formalen Spezifikation ihrer Vorbedingungen und Effekte automatisiert miteinander kombiniert werden können. Management-Planlets fungieren dadurch als komponierbare Managementbausteine, die alle technischen Anweisungen enthalten, um in Kombination die deklarativ modellierten Managementaufgaben in DMMN-Modellen bezüglich der laufenden Anwendung auszuführen. Zudem wird das Konzept der *generischen Management-Planlets* vorgestellt, um die Entwicklung neuer Managementlogik zu vereinfachen.

1.2.4 Automatisierte Anwendung von Managementmustern

Managementaufgaben können mittels DMMN abstrakt modelliert und durch die Orchestrierung von Management-Planlets ausgeführt werden. Technisches Managementwissen kann durch diese Konzepte wiederverwendbar beschrieben und vollautomatisiert für individuelle Anwendungen ausgeführt werden. Die Modellierung der Managementziele in Form von DMMN-Modellen wird in den bisherigen Forschungsbeiträgen manuell durchgeführt, wodurch Managementwissen auf konzeptioneller Ebene von der bisherigen Automatisierung nicht unterstützt wird. Durch das weitverbreitete Konzept der Muster kann bewährtes konzeptionelles Managementwissen zwar dokumentiert [FLR+13; FLR+14; JPCL14], jedoch nicht automatisiert angewendet werden: In der Praxis muss dieses Wissen aufgrund fehlender Automatisierungsmöglichkeiten manuell auf konkrete Anwendungsfälle übertragen werden. Angesichts der enormen technischen Komplexität des Anwendungsmanagements verhindert diese manuelle Vorgehensweise allerdings den effizienten Einsatz des Musterkonzepts. Das Ziel des vierten Forschungsbeitrags ist daher die automatisierte Anwendung von Managementmustern zu ermöglichen, um bewährtes Managementwissen unabhängig von individuellen Anwendungen generisch zu automatisieren.

Forschungsbeitrag 4 (Automatisierte Anwendung von Managementmustern). Der vierte Forschungsbeitrag stellt ein Konzept zur Automatisierung bewährten Managementwissens vor, welches in Form von Mustern textuell dokumentiert ist. Dazu werden *automatisierte Managementmuster* eingeführt, die generisch auf individuelle IT-Systeme angewendet werden können und die Lösung eines Musters automatisiert in Form deklarativer DMMN-Modelle spezifizieren. Dabei werden unterschiedliche Abstraktionshöhen von Managementmustern diskutiert und diese explizit durch das vorgestellte Konzept unterstützt: Automatisierte Managementmuster ermöglichen sowohl die Automatisierung abstrakter Managementexpertise, die anschließend manuell für einen individuellen Anwendungsfall verfeinert wird, als auch die Erfassung spezifischen Wissens, wodurch die Musteranwendung vollständig automatisiert werden kann.

1.2.5 Kontextsensitive Analyse deklarativer Managementaufgaben

Durch die Einführung der deklarativen Sprache DMMN in [Forschungsbeitrag 2](#) und deren Verwendung zur Modellierung deklarativer Managementziele können Managementaufgaben auf hoher Abstraktionsebene sowohl manuell als auch durch automatisierte Managementmuster spezifiziert werden. Dabei können entscheidende Beziehungen zwischen Komponenten, die vom Modellierer aufgrund von Unkenntnis bzw. vom angewandten Muster durch dessen hohe Abstraktion nicht berücksichtigt wurden, zu DMMN-Modellen führen, welche die Anwendung aufgrund nicht beachteter Abhängigkeiten in einen fehlerhaften Zustand führen. Durch die strukturelle Beschreibung der Anwendung inklusive deren Zustand sind die Aufgaben in DMMN-Modellen in ihrem jeweiligen Ausführungskontext modelliert, d. h. in Abhängigkeit der direkt betroffenen Komponenten sowie deren Relationen. In diesem Beitrag wird ein musterbasiertes Verfahren vorgestellt, das diese Informationen zur kontextsensitiven Analyse deklarierter Managementaufgaben verwendet, um mögliche Fehler in DMMN-Modellen vor deren Ausführung sowohl (i) zu identifizieren als auch (ii) durch automatische Adaption des fehlerhaften Modells zu beheben.

Forschungsbeitrag 5 (Kontextsensitive Analyse deklarativer Managementaufgaben). Basierend auf dem in [Forschungsbeitrag 4](#) eingeführten Konzept der automatisierten Managementmuster stellt dieser Forschungsbeitrag ein kontextsensitives Analyseverfahren vor, das die (i) Erkennung sowie die (ii) Korrektur fehlerhaft oder unvollständig modellierter deklarativer Managementmodelle automatisiert. Das Verfahren unterstützt den Modellierer bei der korrekten Erstellung von DMMN-Modellen, indem *automatisierte Analysemuster* Modellierungsfehler sowie unbeachtete Abhängigkeiten zwischen Managementaufgaben, Komponenten und deren Relationen in ihrem jeweiligen Kontext automatisiert erkennen und durch Modifikation des DMMN-Modells beheben. Dadurch kann domänenspezifisches Expertenwissen bezüglich verschiedener Komponententypen und Kombinationen unabhängig von einzelnen Anwendungen generisch erfasst und automatisiert zur präventiven Fehlererkennung angewendet werden.

1.2.6 Architektur und prototypische Implementierung

Um die praktische Umsetzbarkeit der vorgestellten Forschungsbeiträge zu validieren, wird als sechster Beitrag die (i) Architektur und (ii) prototypische Implementierung eines Management-Frameworks vorgestellt. Das System ermöglicht die automatisierte Anwendung der PALMA-Methode aus Forschungsbeitrag 1 zum Management laufender Anwendungen inklusive aller anderen Forschungsbeiträge, welche zur Umsetzung der Methode erforderlich sind.

Forschungsbeitrag 6 (Musterbasiertes Anwendungsmanagement-Framework – Architektur und prototypische Implementierung). In diesem Forschungsbeitrag wird die Architektur eines musterbasierten Management-Frameworks vorgestellt, welche in Form einer prototypischen Implementierung zur praktischen Validierung der Forschungsbeiträge der vorliegenden Arbeit eingesetzt wird. Darin enthalten sind folgende Komponenten: (i) Ein Modellierungswerkzeug für DMMN-Modelle, (ii) ein Anwendungsassistent für automatisierte Managementmuster und (iii) automatisierte Analysemuster sowie (iv) ein vollautomatisierter Plangenerator, um DMMN-Modelle in ausführbare BPEL-Workflows zu transformieren. Die einzelnen Komponenten werden durch das integrierte Gesamtsystem namens *PALMA-Framework* vereint, um die PALMA-Methode vollständig zu automatisieren.

1.3 Veröffentlichungen im Rahmen der Arbeit

Im Rahmen dieses Promotionsvorhabens sind folgende begutachtete („*peer reviewed*“) Publikationen entstanden, welche im Rahmen von Zeitschriften, Konferenzen und Workshops veröffentlicht wurden. Die Publikationen sind dabei gemäß ihrer Relevanz für die vorliegende Arbeit geordnet.

1. U. Breitenbücher, T. Binz, O. Kopp und F. Leymann. „Pattern-based Runtime Management of Composite Cloud Applications“. In: *Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013)*. SciTePress, Mai 2013, S. 475–482

2. U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und J. Wettinger. „Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies“. In: *On the Move to Meaningful Internet Systems: OTM 2013 Conferences (CoopIS 2013)*. Springer, Sep. 2013, S. 130–148
3. U. Breitenbücher, T. Binz, O. Kopp und F. Leymann. „Automating Cloud Application Management Using Management Idioms“. In: *Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014)*. Xpert Publishing Services, Mai 2014, S. 60–69
4. U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und M. Wieland. „Context-Aware Cloud Application Management“. In: *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014)*. SciTePress, Apr. 2014, S. 499–509
5. U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und M. Wieland. „Context-Aware Provisioning and Management of Cloud Applications“. In: *Cloud Computing and Services Science. Communications in Computer and Information Science*. Springer, 2015, S. 151–168
6. U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und D. Schumm. „Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA“. In: *On the Move to Meaningful Internet Systems: OTM 2012 (CoopIS 2012)*. Springer, Sep. 2012, S. 416–424
7. U. Breitenbücher, T. Binz und F. Leymann. „A Method to Automate Cloud Application Management Patterns“. In: *Proceedings of the Eighth International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2014)*. Xpert Publishing Services, Aug. 2014, S. 140–145
8. U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann und J. Wettinger. „Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA“. In: *International Conference on Cloud Engineering (IC2E 2014)*. IEEE, März 2014, S. 87–96

9. U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und M. Wieland. „Policy-Aware Provisioning of Cloud Applications“. In: *Proceedings of the Seventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2013)*. Xpert Publishing Services, Aug. 2013, S. 86–95
10. U. Breitenbücher, T. Binz, C. Fehling, O. Kopp, F. Leymann und M. Wieland. „Policy-Aware Provisioning and Management of Cloud Applications“. In: *International Journal On Advances in Security* 7.1&2 (2014)
11. U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und J. Wettinger. „A Modeling Concept to Integrate Declarative and Imperative Cloud Application Provisioning Technologies“. In: *Proceedings of the 5th International Conference on Cloud Computing and Services Science (CLOSER 2015)*. SciTePress, Mai 2015, S. 487–496

1.4 Aufbau der Arbeit

Zunächst werden in [Kapitel 2](#) die Grundlagen des Anwendungsmanagements vorgestellt sowie verwandte Arbeiten diskutiert. Anschließend wird in [Kapitel 3](#) die PALMA-Methode als erster Forschungsbeitrag vorgestellt. Zur deklarativen Beschreibung von Managementaufgaben wird in [Kapitel 4](#) die Modellierungssprache DMMN eingeführt. Da DMMN-Modelle nicht direkt ausführbar sind, wird in [Kapitel 5](#) das Konzept der Management-Planlets eingeführt, um DMMN-Modelle in ausführbare Managementpläne zu transformieren. [Kapitel 6](#) stellt die Automatisierung von Managementmustern vor, wodurch die automatische Generierung von DMMN Modellen auf Basis bewährten Managementwissens ermöglicht wird. Um die korrekte Modellierung auszuführender Managementaufgaben zu gewährleisten, wird in [Kapitel 7](#) das musterbasierte Konzept zur kontextsensitiven Analyse von Managementaufgaben vorgestellt. Die Architektur und prototypische Implementierung des PALMA-Frameworks wird in [Kapitel 8](#) präsentiert. Abschließend fasst [Kapitel 9](#) die Forschungsbeiträge zusammen und gibt einen Ausblick auf geplante zukünftige Forschungsarbeiten.

KAPITEL 2

GRUNDLAGEN DES ANWENDUNGSMANAGEMENTS UND VERWANDTE ARBEITEN

In diesem Kapitel wird der gegenwärtige Stand existierender Arbeiten in Wissenschaft und Technik hinsichtlich deren Eignung zur musterbasierten Automatisierung des Anwendungsmanagements diskutiert sowie die grundlegende Terminologie definiert. Dazu wird zunächst ein Überblick über Metamodelle zur Beschreibung von Anwendungen gegeben und deren Eignung für das Anwendungsmanagement evaluiert. Anschließend werden verschiedene Managementparadigmen, -konzepte und -technologien diskutiert. Den Abschluss des Kapitels bildet ein Überblick über Muster in der Informationstechnologie, die Musterautomatisierung in anderen IT-Domänen sowie einer Analyse, welche Probleme bei der Anwendung von Managementmustern in der Praxis auftreten und welche Herausforderung deren Automatisierung darstellt.

2.1 Modelle zur Beschreibung von Anwendungen

In diesem Abschnitt werden verschiedene Ansätze und Perspektiven zur Beschreibung von Anwendungen vorgestellt und grundlegende Begriffe definiert.

2.1.1 Architekturen

Architekturmodelle werden typischerweise zur Konzeption, Dokumentation und Kommunikation einer abstrakten Sicht auf ein System eingesetzt [Bin15]. In der Literatur gibt es eine Vielzahl unterschiedlicher Definitionen des Begriffs. Insbesondere der Begriff *Softwarearchitektur* erfährt unterschiedliche Interpretationen bezüglich technischer Granularität, Umfang und Nähe zur Implementierung [IEE90; GP95; BRJ05; JRvL00; Fow02; BCK03; KBS04; LL07]. Folglich fällt es schwer, eine einvernehmliche Definition auf Basis gegenwärtiger Literatur zu finden. Fowler [Fow02] identifiziert zwei verbreitete Eigenschaften, welche die meisten Architekturdefinitionen explizit oder implizit gemein haben: Sie beschreiben (i) die höchstmögliche Aufschlüsselung eines Systems in seine Bestandteile sowie (ii) Entscheidungen, die nachträglich schwer zu ändern sind. Für die vorliegende Arbeit wird die Definition des IEEE-Standards 1471 (2000) [IEE00] verwendet, welche aufgrund der Abstraktion von Software für die Beschreibung verschiedenster Systeme geeignet ist, in denen beispielsweise auch Hardwareaspekte eine Rolle spielen - etwa wenn die Mobilität involvierter Geräte wie Smartphones die Systemarchitektur beeinflusst:

Definition 1 (Architektur - informell). Eine Architektur beschreibt die fundamentale Organisation eines Systems, verkörpert durch seine Komponenten, deren Beziehungen untereinander und zur Umgebung sowie die Prinzipien, welche zu dessen Entwurf und Evolution geführt haben [IEE00]. ■

Zur Beschreibung von Architekturen existieren verschiedene Modellierungssprachen (engl. „*Architecture Description Languages*“ (ADL) [GMW97]), beispielsweise die *Fundamental Modeling Concepts* (FMC) [KW03; Tab05], *Acme* [GMW97; GMW00] und *ArchiMate* [HP12].

2.1.2 Anwendungstopologien

Eine *Anwendungstopologie* umfasst den strukturellen Aufbau einer Anwendung in Form der involvierten Komponenten sowie deren technische Relationen untereinander. Anwendungstopologien können durch formale, graphbasierte *Topologiemodelle* beschrieben werden, in denen Knoten die Komponenten der Anwendung repräsentieren, Kanten deren Relationen [BBKL14b; AGLW14]. Eine Relation steht dabei für eine Abhängigkeit zwischen genau zwei Komponenten, beispielsweise, um zu modellieren, dass eine Komponente auf einer anderen betrieben wird. Die Menge aller in einem Topologiemodell repräsentierten Komponenten und Relationen wird in dieser Arbeit als *Topologieelemente* bezeichnet. Durch Angabe des jeweiligen Komponenten- bzw. Relationentyps wird typischerweise die Semantik des jeweiligen Topologieelements spezifiziert.

Ein Topologiemodell umfasst neben dem strukturellen Aufbau der Anwendung je nach Topologiesprache weitere Informationen, wie beispielsweise Eigenschaften, zugehörige Artefakte oder verfügbare Operationen, die von den repräsentierten Komponenten und Relationen angeboten werden. Topologiemodelle können somit zur Beschreibung der technischen Modularisierung einer Anwendung eingesetzt werden, welche z. B. die Grundlage zur Entwicklung hochverfügbarer Systeme darstellt [Gra86]. Die *Topology and Orchestration Specification for Cloud Applications (TOSCA)* [OAS13b; OAS13a] ist ein von OASIS veröffentlichter Standard zur Beschreibung von Anwendungstopologien.

Im Gegensatz zu Architekturmodellen, die eine Anwendung auf hoher Abstraktion zur Dokumentation der Systemsicht beschreiben, stellen Topologiemodelle eine technisch detaillierte *Deployment-Sicht* der Anwendungsstruktur dar. Sie ermöglichen dadurch die deklarative Beschreibung der gewünschten Anwendungsbereitstellung [OAS13a], die aufgrund der detaillierten technischen Beschreibung vollautomatisiert durchgeführt werden kann. Beispielsweise indem ein ausführbares Workflow-Modell generiert wird, deren Ausführung die modellierte Anwendung automatisiert provisioniert [Mie10; BBK+14a]. Topologiemodelle werden daher häufig *präskriptiv* verwendet, um die Struktur und technischen Details einer bereitzustellenden Anwendung zu spezifizieren.

Die Modellart ermöglicht aber auch die Beschreibung einer Anwendung aus der *deskriptiven* Perspektive, um die technische Struktur einer existierenden Anwendung zu dokumentieren. Soll eine Anwendungsinstanz jedoch detailliert inklusive aller Laufzeitinformationen als formales Modell erfasst werden, eignen sich dazu *Anwendungsinstanzmodelle*, die häufig auch in Form von Topologien strukturiert sind (siehe [Abschnitt 2.1.3](#)). In Anlehnung an Binz [[Bin15](#)] werden Anwendungstopologiemodelle wie folgt informell definiert:

Definition 2 (Anwendungstopologiemoell - informell). Ein Anwendungstopologiemoell ist ein formales, graphbasiertes Modell einer Anwendung, das alle aus Managementsicht relevanten Komponenten der Anwendung (Knoten) sowie deren Relationen (Kanten) auf technischer Ebene beschreibt. ■

2.1.3 Anwendungsinstanzmodelle

Ein *Anwendungsinstanzmodell* beschreibt den Zustand und die Konfiguration einer laufenden Anwendung. Instanzmodelle sind von fundamentaler Bedeutung für die Durchführung von Änderungen an der IT-Landschaft, z. B. wenn eine Komponente in eine Cloud-Umgebung migriert werden soll [[BBKL14a](#); [Bin15](#)]. Ist kein detailliertes Wissen über die laufende Anwendung in Form solcher Instanzmodelle vorhanden, führen IT-Änderungen häufig zu Fehlern und Systemausfällen [[OGP03](#); [Opp03](#); [BBK+14b](#); [BBK+15b](#)].

Instanzmodelle können auf verschiedene Arten formalisiert werden. Binz et al. [[BFL+12](#); [BBKL14a](#); [Bin15](#)] stellen mit den *Enterprise Topologie Graphen* ein Metamodell vor, um Instanzmodelle auf Basis von Anwendungstopologien zu modellieren. Die enthaltenen Anwendungskomponenten und Relationen werden mit zusätzlichen Instanzinformationen angereichert, um die jeweiligen Zustände und Konfigurationen zu beschreiben. Der Vorteil dieser topologiebasierten Repräsentation ist die Möglichkeit, Abhängigkeiten zwischen Komponenten explizit zu erfassen. Frank et al. [[FHK+09](#)] stellen mit der *IT Domain-Specific Modeling Language (ITML)* eine Beschreibungssprache vor, welche unter anderem die Modellierung von Beziehungen zwischen Geschäftsprozessen und

IT-Systemen inklusive deren Software- und Hardwarekomponenten ermöglicht. Die Sprache erlaubt auch Instanzinformationen an die Elemente der Sprache zu annotieren, beispielsweise, um die Anzahl aktueller Instanzen eines Geschäftsprozesses zu erfassen. Beide Sprachen ermöglichen eine umfangreiche Analyse einer Anwendung, da diese in ihrer Gesamtheit und Einbettung in umliegende Systeme erfasst werden kann. Ein weiteres verbreitetes Konzept zur Erfassung und Verwaltung von Instanzinformationen stellen die sogenannten *Konfigurationsmanagementdatenbanken* dar (engl. „*Configuration Management Databases (CMDB)*“ [Can07]). Diese Datenbanksysteme speichern alle relevanten Informationen über die Komponenten einer Anwendung, welche verwaltet werden müssen, um diese bereitstellen, betreiben und managen zu können. Sie bilden somit in vielen Managementprozessen den zentralen Informationspunkt [MSB+07; vJBN+10]. Die Schemata der Datenbanksysteme sind dabei je nach Implementierung unterschiedlich und beschreiben Ressourcen häufig in objektorientierter Form, welche in einigen Systemen vom *Common Information Model (CIM)* Standard [DMT10] abgeleitet wurde [KS09; HK10b].

Instanzmodelle werden entweder manuell oder automatisiert erstellt. Die manuelle Erstellung wird händisch von einem Administrator durchgeführt und ist unter anderem aufgrund des enormen Zeitbedarfs ineffizient [MDW+00; MSB+07]. Zur automatisierten Erstellung von Instanzmodellen wurden verschiedene Ansätze und Werkzeuge entwickelt. Machiraju et al. [MDW+00] stellen beispielsweise einen automatisierten Ansatz vor, der Instanzinformationen in Form von UML-Klassendiagrammen [OMG07] abbildet. Binz et al. [BBKL13a; Bin15] entwickelten ein Crawling-Werkzeug, das Anwendungsinstanzmodelle vollautomatisiert in Form von Enterprise Topologie Graphen erstellt. Konfigurationsmanagementdatenbanken werden typischerweise mit Werkzeugen kombiniert oder stellen selbst Mechanismen zur Verfügung, um Instanzdaten automatisch zu erfassen und zu aktualisieren [MSB+07]. Keller und Kar [KK01] stellen eine Systemarchitektur vor, um Abhängigkeiten¹ zwischen Anwendungskomponenten auf Anfrage mittels einer *Dependency Query Facility* zu berechnen.

¹In einer früheren Arbeit [KBK00] stellen die Autoren zudem eine Klassifikation verschiedener Abhängigkeitseigenschaften zwischen Anwendungskomponenten vor.

Gleichermaßen wie Anwendungstopologiemodelle können Instanzmodelle sowohl deskriptiv als auch präskriptiv verwendet werden. Enterprise Topologie Graphen werden beispielsweise dazu eingesetzt, den Istzustand laufender Anwendungen in einer IT-Landschaft deskriptiv zu beschreiben, um Anwendungskomponenten in eine Cloud-Umgebung zu migrieren. Andererseits werden Instanzmodelle auch präskriptiv verwendet, um den Sollzustand bzw. eine gewünschte Zielkonfiguration einer Anwendung zu spezifizieren, welche anschließend von einem Managementsystem durch Ausführung von Managementoperationen erzwungen wird. Diese Form der Nutzung spielt eine fundamentale Rolle im deklarativen Managementparadigma und wird im Rahmen existierender Technologien in [Abschnitt 2.3.3](#) detailliert analysiert und diskutiert.

In der vorliegenden Arbeit werden topologiebasierte Instanzmodelle zur Erfassung des Istzustandes einer Anwendung gewählt, da diese Modellart die Beschreibung aller für das Anwendungsmanagement relevanten Informationen ermöglicht und diese in strukturierter Form dokumentiert: Sie ermöglichen sowohl die Beschreibung aller Anwendungskomponenten und deren Relationen sowie der jeweiligen Instanzinformationen. Dadurch können Managementaufgaben direkt mit den Komponenten und Relationen assoziiert werden, welche durch die auszuführenden Aufgaben betroffen sind. Zudem ermöglicht diese Form der Informationserfassung eine intuitive graphische Visualisierung, beispielsweise durch Verwendung von Sprachen wie VINO4TOSCA [\[BBK+12\]](#), die ein Metamodell zur graphischen Darstellung von Topologien definieren. Das Konzept der topologiebasierten Instanzmodelle dient bei der Umsetzung der PALMA-Methode als Grundlage zur Definition der deklarativen Sprache DMMN, welche die deklarative Spezifikation von Managementaufgaben direkt an den betroffenen Komponenten und Relationen ermöglicht. In Anlehnung an [Definition 2](#) werden Anwendungsinstanzmodelle daher wie folgt definiert:

Definition 3 (Anwendungsinstanzmodell - informell). Ein Anwendungsinstanzmodell ist ein Topologiemodell, das alle aus Managementsicht relevanten Komponenten einer Anwendung (Knoten), deren Relationen untereinander (Kanten) sowie deren Zustände und Konfigurationen beschreibt. ■

2.1.4 Sprachen zur Komposition und Bereitstellung von Anwendungen

Zur Modellierung von Anwendungen wurden verschiedene Metamodelle und domänenspezifische Sprachen (DSLs) [Hud98; MHS05] entwickelt, die unter anderem zur Komposition von Anwendungen sowie zu deren Bereitstellung eingesetzt werden. In diesem Abschnitt werden die wichtigsten Sprachen vorgestellt und auf Eignung bezüglich des Einsatzes bzw. einer Erweiterung für das Anwendungsmanagement untersucht. Eine Zusammenfassung dieses Abschnitts sowie eine abschließende Diskussion wird in [Abschnitt 2.1.4.2](#) gegeben.

Die *Service Component Architecture (SCA)* [OAS07a] ist eine Menge an standardisierten Spezifikationen, die ein Metamodell zur Entwicklung dienstorientierter Anwendungen definieren [FS06]. SCA ermöglicht die Beschreibung und Implementierung von Komponenten und deren Schnittstellen, sowie die Modellierung von Kompositionen dieser Komponenten, um eine höherwertige Geschäftslogik zu realisieren. Zur Implementierung können verschiedene Sprachen eingesetzt werden, beispielsweise BPEL [OAS09] oder Java [PD06]. Als Grundlage des Anwendungsmanagements eignen sich SCA-Modelle nur bedingt, da eine detaillierte Darstellung der Anwendungsstruktur inklusive individueller, semantisch definierter Relationen zwischen Anwendungskomponenten nicht möglich ist.

Aufgrund der im Cloud Computing erforderlichen Automatisierung der Bereitstellung und des Managements von Anwendungen wurden in den vergangenen Jahren zahlreiche Modellierungssprachen entwickelt. Das MOCCA-Metamodell¹ [LFM+11] definiert beispielsweise eine generische Sprache zur Beschreibung von Anwendungen aus verschiedenen Perspektiven und auf unterschiedlichen Granularitätsebenen. MOCCA ermöglicht sowohl die Beschreibung der Anwendungsarchitektur sowie die Modellierung technisch detaillierter Topologiemodelle, die zur automatisierten Bereitstellung der Anwendung durch das *Cafe-Framework* [MUL09; Mie10] eingesetzt werden können. Das Cafe-Framework unterstützt zudem das ursprüngliche Cafe-Metamodell, das auf ähnlichen Konzepten basiert [MUL09; Mie10].

¹Move to Clouds for Composite Applications

Die *Topology and Orchestration Specification for Cloud Applications (TOSCA)* ist ein Standard [OAS13b; OAS13a], der als Erweiterung des MOCCA-Metamodells gesehen werden kann. Die Spezifikation definiert ein generisches, graphbasiertes Metamodell zur Modellierung portabler Topologiemodelle und ermöglicht die Beschreibung typisierter Komponenten, deren Relationen untereinander sowie deren Eigenschaften [BBL12; BBKL14b]. Zusätzlich können, im Gegensatz zu MOCCA, auch Managementschnittstellen, welche von Komponenten und Relationen angeboten werden, beschrieben werden. Diese können von einer Laufzeitumgebung oder expliziten Managementprozessmodellen, die in TOSCA *Managementpläne* (engl. „*Management Plans*“ [OAS13b]) genannt werden, zur Bereitstellung bzw. zum Management der Anwendung aufgerufen werden. Aufgrund der umfassenden Beschreibungsmöglichkeiten einer Anwendungstopologie inklusive deren Schnittstellen und Eigenschaften können TOSCA-Modelle zur deklarativen Modellierung der Bereitstellung von Anwendungen eingesetzt werden [OAS13a; BBK+14a]. Das Konzept der Managementpläne ermöglicht zudem die Automatisierung der Managementlogik und unterstützt dadurch auch die imperative Modellierung von Bereitstellungsprozessen komplexer Anwendungen. Die deklarative Modellierungssprache DMMN, welche in der vorliegenden Arbeit vorgestellt wird, verwendet ähnliche Konzepte zur graphbasierten Modellierung von Anwendungen auf Basis typisierter Komponenten und Relationen sowie zur Modellierung und Implementierung von Managementschnittstellen. DMMN erweitert die TOSCA-Konzepte jedoch signifikant um die Möglichkeit auszuführende Managementaufgaben deklarativ zu modellieren. Zudem werden verschiedene Einschränkungen von TOSCA in DMMN aufgehoben. Die visuelle Notation VINO4TOSCA [BBK+12], die zur graphischen Modellierung von Anwendungstopologien eingesetzt werden kann, stellt zudem die Grundlage der grafischen Notation von DMMN dar. Eine detaillierte Abgrenzung zu TOSCA wird in [Abschnitt 4.6](#) vorgestellt.

CloudML [FRC+13; FSR+14], früher *Pim4Cloud* [BMM12], ist eine Sprache zur Modellierung der Bereitstellung von Multi-Cloud-Anwendungen. Die Sprache ist im Gegensatz zu TOSCA weniger generisch, was sich beispielsweise durch nativ definierte Metaklassen für virtuelle Maschinen und Anbieter (engl.

„provider“) bemerkbar macht [FSR+14]. Generell können komponentenbasierte Anwendungen beschrieben werden, die verschiedene Infrastruktur- und Plattformdienste zum Betrieb verwenden. Zudem können Relationen zwischen Komponenten spezifiziert werden, beispielsweise, dass zwei Komponenten über spezifizierte Ports miteinander kommunizieren. Die Sprache ist dabei prinzipiell deklarativ, ermöglicht aber die imperative Spezifikation auszuführender Skripte, um z. B. eine Komponente zu installieren [FSR+14]. Die Ausführung von CloudML-Modellen wird durch das *CloudMF Environment* [FCR+13; FSR+14] ermöglicht. Diese Managementumgebung unterstützt zudem Änderungen an der laufenden Anwendung gemäß manuell getätigter Veränderungen am zugehörigen CloudML-Modell durchzuführen, indem der aktuelle Istzustand der Anwendung mit dem neu spezifizierten Sollzustand verglichen wird und auszuführende Aktionen abgeleitet und ausgeführt werden [FSR+14]. Die Umgebung kann daher auch für Managementaufgaben verwendet werden, beispielsweise um eine Anwendungskomponente zu skalieren [FBR+14]. Aufgrund der eingeschränkteren Generizität im Vergleich zu TOSCA und dem Fokus auf *zustandsverändernde* Managementaufgaben eignen sich die CloudML-Konzepte nicht für ein flexibel erweiterbares Anwendungsmanagement, da *zustandserhaltende* Aufgaben, wie beispielsweise der Export von Daten aus einer Datenbank, nicht modelliert werden können. Eine ausführliche Diskussion und Abgrenzung dieser beiden Typen von Managementaufgaben wird in [Abschnitt 2.3.1](#) geführt.

Unter demselben Akronym stellen Gonçalves et al. [GES+11] eine Sprache vor, welche die Beschreibung von Cloud-Ressourcen mit Fokus auf die Infrastruktur ermöglicht und dadurch nicht beliebige Komponententypen abbilden kann. Dieser und andere infrastrukturzentrischen Ansätze, wie beispielsweise *Uni4Cloud* [SM11], *RESERVOIR-ADL* [CEM+10], das *Open Virtualization Format (OVF)* [DMT09], dessen Erweiterung für Cloud-Umgebungen von Galán et al. [GSR+09], *Cloud#* [LZ11], das *Virtual Appliances Model* [KEK+09], das *Heat Orchestration Template (HOT)* [Ope15], *VXDL* [KVC09], *System Pattern Descriptions* [LSS+13] sowie der Ansatz von Kirschnick et al. [KAG+12], eignen sich daher nicht zur Modellierung von Managementaufgaben für modularisierte Anwendungen, die z. B. auch Plattform- oder Softwaredienste involvieren.

UML 2.0 definiert mit den *Deployment Diagrams* [RJB10] eine Sprache, welche diese Form der Modellierung modularisierter Anwendungen ermöglicht. Deployment-Diagramme erlauben die Modellierung von Komponenten und deren Beziehungen zueinander und sind daher mit Anwendungstopologien vergleichbar. Zudem ermöglichen standardisierte Stereotypen die Spezifikation der Semantik von Komponenten und Artefakten, beispielsweise, indem ein Artefakt als *executable* stereotypisiert wird. Der grundsätzliche Aufbau von DMMN ist aufgrund der Nähe zu Topologien ähnlich zu dieser Diagrammform, erweitert diese jedoch um die Möglichkeiten, Managementschnittstellen zu spezifizieren und zu implementieren sowie Managementaufgaben zu deklarieren. Andere Diagrammtypen der UML, die zur strukturellen Beschreibung eingesetzt werden, beispielsweise *Klassendiagramme* [RJB10], eignen sich nur bedingt zur Modellierung von Bereitstellungen, da diese zur Beschreibung der Interna von Komponenten bzw. deren Interaktion entwickelt wurden.

Die hohe Generizität von UML Deployment-Diagrammen erschwert die Modellierung Cloud-basierter Anwendungen, die von konkreten Anbietern bereitgestellt werden sollen, da diese durch die generischen Konzepte der UML nachgebildet werden müssen. Um die Verwendung existierender Cloud-Angebote bei der Modellierung zu unterstützen, stellen Bergmayr et al. [BTN+14] die *Cloud Application Modeling Language (CAML)* vor, die UML-Profile definiert, um Cloud-spezifische Ressourcen und Dienste in UML zu modellieren. Diese Profile definieren UML-Stereotypen für Cloud-Dienste konkreter Anbieter, wie beispielsweise Amazon, und ermöglichen dadurch eine detaillierte Beschreibung der gewünschten Bereitstellung. Da CAML auf UML basiert, ermöglicht die Sprache jedoch nicht die Spezifikation und Implementierung expliziter Managementschnittstellen, Operationen und Managementpläne, wodurch TOSCA bezüglich der Bereitstellung von Anwendungen umfassender ist und auch das Management abdeckt. Die von CAML definierten Profile sind somit komplementär zum vorgestellten DMMN-Metamodell, da dieses, ebenso wie UML und TOSCA, ausschließlich generische Konzepte definiert, jedoch keine konkreten, Cloud-spezifischen Dienste. CAML kann somit als Vorlage zur Definition Cloud-basierter Komponententypen in DMMN eingesetzt werden.

Einen ähnlichen Ansatz zur Erweiterung von UML für die Domäne des Cloud Computings stellt *MULTICLAPP* [GMMC13b] dar. *MULTICLAPP* definiert, gleichermaßen wie CAML, ein UML-Profil zur Modellierung Cloud-basierter Anwendungen, zielt dabei jedoch auf die Portabilität von Multi-Cloud-Anwendungen ab. Das Metamodell sieht daher vor, dass verwendete Cloud-Dienste unabhängig von einem konkreten Anbieter modelliert werden und definiert dazu, im Vergleich zu CAML, abstrakte UML-Stereotypen. Die Implementierung der Anwendungskomponenten folgt diesem Vorgehen und ist somit agnostisch zu individuellen Cloud-Anbietern. *MULTICLAPP* wird von einem Framework [GMMC13a] unterstützt, dass die anbieterunabhängigen Modelle in anbieterspezifische Modelle übersetzt und zusätzlich nötige Adapter generiert, um die Anwendungsimplementierungen für die gewählten Zielumgebungen lauffähig zu machen. Die von *MULTICLAPP* eingeführte Cloud-Abstraktion wird in der vorliegenden Arbeit nicht berücksichtigt: In DMN-Modellen werden die konkreten Cloud-Dienste, Anbieter und Komponenten modelliert, auf Basis deren die jeweilige Anwendung zum Zeitpunkt des Managements betrieben wird.

Nguyen et al. [NLT+11; NLPvH12] führen das Konzept der *Blueprint Templates* ein, das eine uniforme, abstrakte Beschreibung eines Cloud-Dienstes ermöglicht, welche von allen anbieterspezifischen technischen Details abstrahiert und somit agnostisch zu dessen Implementierung ist. Das Konzept hilft Anwendungsentwicklern bei der Auswahl, Anpassung und Komposition beliebiger Cloud-Dienste (IaaS, PaaS, SaaS, etc.) verschiedener Anbieter [NLPvH12]. Ein wesentlicher Bestandteil dieser *Blueprint Templates* stellt die sogenannte *Virtual Architecture* dar, die Abhängigkeiten zwischen den Komponenten und Artefakten einer Anwendung sowie deren Kommunikations- und Deployment-Beziehungen als gerichteten Graphen modelliert. Durch verschiedene Module [Ngu13] werden Cloud-Angebote auf verschiedenen Ebenen definiert, beispielsweise Eigenschaften und Abhängigkeitsbeziehungen von PaaS-Diensten. Der Fokus liegt jedoch auf der anbieterunabhängigen Beschreibung involvierter Ressourcen, Dienste und deren abstrakten Kommunikations- und Abhängigkeitsbeziehungen, wodurch Anwendungsstrukturen zwar modelliert, jedoch nicht in derselben Detailtiefe wie es z. B. TOSCA ermöglicht beschrieben werden können.

Silva et al. [SRC14] stellen eine domänenspezifische Sprache vor, um Cloud-Ressourcen inklusive deren Managementschnittstellen in einem einheitlichen Vokabular zu beschreiben. In ähnlicher Form entwickelten Frey und Hasselbring im Rahmen des *CloudMIG-Ansatzes* [FH10; FH11b] das *Cloud Environment Model (CEM)* [FH11a], das zur Beschreibung von Infrastruktur- und Plattformdiensten von Cloud-Anbietern verwendet werden kann und auf dem *Knowledge Discovery Metamodel (KDM)* [OMG11a] basiert. Beide Metamodelle ermöglichen jedoch nur die Beschreibung einzelner Cloud-Angebote und deren Eigenschaften, jedoch nicht die Modellierung ganzer Anwendungsstrukturen. Somit sind diese Art der Sprachen weder zur Modellierung der Bereitstellung von Anwendungen, noch zur Modellierung von Managementaufgaben geeignet.

2.1.4.1 Grafische Notationen zur Anwendungsmodellierung

Zur grafischen Modellierung von Anwendungen wurden verschiedene domänenspezifische Notationen entwickelt. Bekannte und verbreitete Beispiele zur Modellierung von Architekturen stellen das *Fundamental Modeling Concept (FMC)* [KW03] oder *ArchiMate* [HP12] dar. Auf Ebene der Anwendungstopologien wurde mit *Vino4TOSCA* [BBK+12] eine grafische Notation zur Beschreibung von TOSCA-basierten Modellen vorgestellt. Die grafische Modellierung komponentenbasierter Systeme ermöglicht die Unified Modeling Language [RJB10] durch die sogenannten *Component Diagrams*. Andere Sprachen zur Beschreibung von Cloud-Anwendungen und deren Bereitstellung verwenden häufig eine grafische Syntax, definieren jedoch kein explizites Metamodell für die Notation, wie beispielsweise CloudML [FSR+14].

Der wesentliche Vorteil von domänenspezifischen Notationen gegenüber generellen Konzepten wie ER-Modellen (*Entity Relationship Models* [Che76]) liegt in der Optimierbarkeit von Syntax und Semantik bezüglich der jeweiligen Domäne [FHK+09]. Die Abstinenz einer grafischen Notation für eine domänenspezifische Sprache kann daher schnell zu Akzeptanzproblemen führen, wenn für die DSL aufgrund des Fehlens einer eigenen Notation eine generelle Notation eingesetzt werden muss, die bezüglich der Domäne nicht optimiert wurde und somit nicht ideal geeignet ist. Beispielsweise wurde die fehlende

Definition und Standardisierung einer grafischen Notation für die Workflow-Sprache BPEL nachträglich als Fehler angesehen [Ley10]. Daher wird in der vorliegenden Arbeit mit DMMN sowohl (i) ein domänenspezifisches Metamodell zur deklarativen Beschreibung von Managementaufgaben entwickelt sowie (ii) eine zugehörige Notation definiert, welche auf diesem Metamodell basiert.

Zur Entwicklung kognitiv effektiver grafischer Notationen stellt Moody [Moo09] verschiedene Prinzipien vor, welche die physikalischen Wahrnehmungsfähigkeiten des Menschen explizit berücksichtigen und dadurch zur Optimierung der Lesbarkeit und Kommunikation grafischer Diagramme beitragen. Die Prinzipien dieser *Design Theory* wurde in der vorliegenden Arbeit zur Konzeption der visuellen Syntax von DMMN angewandt (siehe Kapitel 4).

2.1.4.2 Zusammenfassung

Die vorgestellten Modellierungssprachen unterscheiden sich vorwiegend in ihrer Generizität bezüglich (i) modellierbarer Komponententypen und deren Beziehungen zueinander sowie (ii) der Modellierbarkeit managementkritischer Informationen, z. B. Schnittstellen und Eigenschaften von Komponenten und Beziehungen. Ein wesentliches Kriterium, um Anwendungsmanagement verlässlich und fehlerfrei durchzuführen, ist die Erfassung aller relevanten Informationen über die betroffene Anwendung [BBK+14b]. Dies beinhaltet zum einen eine detaillierte Repräsentation der Anwendungsstruktur, zum anderen die explizite Erfassung von Instanzinformationen über die Komponenten und Relationen, wie es durch Anwendungsinstanzmodelle ermöglicht wird (vgl. Abschnitt 2.1.3). Im Gegensatz zu den anderen vorgestellten Sprachen erfüllt der Tosca-Standard diese Kriterien weitestgehend, da dieser neben der Bereitstellung von Anwendungen auch explizit für das Anwendungsmanagement ausgelegt wurde. Er unterstützt jedoch ausschließlich die deklarative Modellierung der Bereitstellung von Anwendungen, nicht die deklarative Modellierung beliebiger Managementaufgaben. In der vorliegenden Arbeit wird mit DMMN eine Sprache vorgestellt, die einige der von Tosca eingeführten Konzepte verwendet und modifiziert. Hauptsächlich werden jedoch fundamentale Erweiterungen hinsichtlich des deklarativen Anwendungsmanagements einführt.

2.2 Anwendungsmanagement

In diesem Abschnitt werden die Grundlagen des Anwendungsmanagements vorgestellt. Dazu wird in [Abschnitt 2.2.1](#) zunächst der generelle Lebenszyklus von IT-Systemen betrachtet. [Abschnitt 2.2.2](#) stellt organisatorische Rahmenwerke des Anwendungsmanagements vor. Die Unterschiede zwischen der manuellen Managementdurchführung sowie dem automatisierten Anwendungsmanagement werden in [Abschnitt 2.2.3](#) diskutiert. Anschließend werden in [Abschnitt 2.3](#) technische Managementtechnologien und -paradigmen vorgestellt sowie deren Eignung zum Management komplexer Anwendungen diskutiert.

2.2.1 Anwendungslebenszyklus

Anwendungen durchlaufen während ihrer Lebenszeit verschiedene Phasen, die häufig als *Lebenszyklus* oder *Lebenslauf* [LL07] bezeichnet werden. Das Management dieser Anwendungslebenszyklen (engl. „*Application Lifecycle Management (ALM)*“) umfasst dabei den gesamten Zeitraum von der initialen Planung, über die Entwicklung der Anwendung, deren Betrieb, Wartung und finalen Außerbetriebnahme [Cha08]. Neben finanziellen und organisatorischen Entscheidungen hinsichtlich der Steuerung und Planung des Vorhabens (engl. „*governance*“), wirken sich die in [Abbildung 2.1](#) vereinfacht dargestellten Phasen *Entwicklung* und *Betrieb* direkt auf die Funktionalität der Anwendung aus.

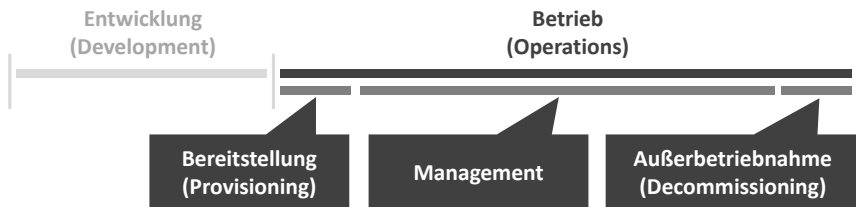


Abbildung 2.1: Entwicklungs- und Betriebsphase des Anwendungslebenszyklus

In der Entwicklungsphase (engl. „*development*“) wird die Anwendung konzipiert, entworfen und implementiert. Dabei werden alle Schritte von der ersten Idee, über die Architektur, bis zur auslieferbaren Software durchlaufen.

In der Betriebsphase (engl. „*operations*“) wird die entwickelte Anwendung zu deren Nutzung betrieben. Dabei kann wiederum zwischen drei Subphasen unterschieden werden [Cha08]: (ii.a) Bereitstellung, (ii.b) Management und (ii.c) Außerbetriebnahme. In der Bereitstellungsphase (engl. „*provisioning*“) wird die Anwendung in Betrieb genommen, d. h. sie wird in der Zielumgebung installiert, konfiguriert und gestartet. Daraufhin beginnt die Phase des Managements, unter welchem in dieser Arbeit jegliche Form von Änderungen an der betriebenen Anwendung verstanden wird. Darunter fallen Wartungsarbeiten, die gemäß Ludewig und Lichter „*nicht von Beginn der Software-Entwicklung geplant waren oder hätten geplant werden können und die unmittelbare Auswirkungen auf den Benutzer der Software haben*“ [LO04; LL07]. Das Anwendungsmanagement umfasst somit alle Aufgaben zur adaptiven Wartung, korrektiven Wartung und zur verbessernden Wartung [LL07] (engl. „*adaptive maintenance*“, „*corrective maintenance*“, „*perfective maintenance*“ [IEE90]), beispielsweise das Einspielen von Updates, die Migration von Anwendungskomponenten oder das Reengineering der Architektur. Zusätzlich umfasst es jegliche Form von *Change Management* [Lie06], *Operations Management* und *Configuration Management* [Hew13; DJV10]. Gemäß Sturm und Bumpus [SB98] ist allgemein die Steuerung, Administration und Kontrolle der Benutzung von Anwendungssystemen sowie deren Funktionen und Eigenschaften Teil des Anwendungsmanagements. Die Betriebsphase sowie der gesamte Anwendungslebenszyklus enden mit der finalen Außerbetriebnahme der Anwendung (engl. „*decomissioning*“).

Diese Dissertation fokussiert sich auf die Betriebsphase des Anwendungslebenszyklus und stellt dazu Konzepte vor, um die Bereitstellung, das Management sowie die Außerbetriebnahme von Anwendungen zu automatisieren. Die in diesem Abschnitt beschriebene zeitliche Abfolge der Phasen dient dabei ausschließlich der wissenschaftlichen Einordnung der Forschungsbeiträge, die auch in modernen Ansätzen wie *Continuous Delivery* [HF10] oder DevOps [Hüt12], welche Iterationen bzw. modifizierte Phasen definieren, einsetzbar sind.

2.2.2 IT-Service Management

In diesem Abschnitt werden organisatorische Rahmenwerke zum Management von Informationstechnik vorgestellt (engl. „*IT-Service-Management*“ [Kle13]).

Die *IT Infrastructure Library (ITIL)* [APM16] ist ein weitverbreitetes Referenzmodell für IT-Prozesse, das aus einer Sammlung an Praktiken zur Umsetzung des IT-Service-Managements besteht. Ein Kerngebiet von ITIL stellt das *Change Management* [Lie06] dar, das Richtlinien zur Ausführung von Änderungen an IT-Infrastrukturen vorgibt. Zur Spezifikation durchzuführender Änderungen werden sogenannte *Requests for Change (RFC)* eingesetzt, die deklarativ beschreiben, welche Änderungen an den Komponenten der Infrastruktur durchgeführt werden sollen. Diese spezifizieren keinerlei technische Ausführungsdetails und werden entweder manuell von Administratoren oder automatisiert durch Managementsysteme verarbeitet bzw. in sogenannte *Change Plans* übersetzt [WBL+09], beispielsweise unterstützt durch die Ansätze von Cordeiro et al. [dCMA+08; dCMD+08; dCMA+09] (siehe Abschnitt 2.3.4.3). Die in der vorliegenden Arbeit eingeführte deklarative Modellierungssprache DMMN kann generell zur Modellierung von RFCs eingesetzt werden, wodurch die Forschungsbeiträge zur Realisierung und Automatisierung diesbezüglicher ITIL-Prozesse im Rahmen des Change Managements eingesetzt werden können.

COBIT [ISA16] ist ein IT-Managementrahmenwerk [FHK+09], das die ganzheitliche IT-Steuerung (engl. „*IT-Governance*“) durch Dokumentation gängiger, generell akzeptierter Praktiken unterstützt und den gesamten Lebenszyklus von IT-Systemen und IT-Dienstleistungen beschreibt [Koo06].

Open Services for Lifecycle Collaboration (OSLC) standardisiert Spezifikationen zur Integration der Anwendungsentwicklung und zum Anwendungsbetrieb. Darunter fallen beispielsweise Spezifikationen zum Application Lifecycle Management (ALM) sowie zum Change Management [OAS08].

Diese Rahmenwerke und Spezifikationen sind komplementär zu den Forschungsbeiträgen der vorliegenden Arbeit zu verstehen, um die PALMA-Methode in die organisatorischen Managementprozesse eines Unternehmens einzubinden.

2.2.3 Managementdurchführung

Nachdem im vorherigen Abschnitt organisatorische Rahmenwerke zur Organisation von Managementprozessen vorgestellt wurden, diskutiert dieser Abschnitt deren technische Durchführung. Generell wird bei der Durchführung von Managementaufgaben zwischen der (i) manuellen und (ii) automatisierten Vorgehensweise unterschieden, wobei der Automatisierungsgrad von semi-automatisiert bis hin zur vollständigen Automatisierung reicht. Im Folgenden werden die unterschiedlichen Vorgehensweisen sowie deren Vor- und Nachteile evaluiert und auf Eignung hinsichtlich verschiedener Konstellationen diskutiert.

2.2.3.1 Manuelle Managementdurchführung

In vielen Organisationen ist die Bereitstellung, das Management sowie die Außerbetriebnahme von Anwendungen ein manueller Prozess, der von Systemadministratoren mithilfe technischer Werkzeuge, wie Kommandozeilen, schrittweise durchgeführt wird [HF10; FLRS12]. Zur Auslieferung einer neuen Software-Release werden beispielsweise händisch die benötigten (virtuellen) Maschinen bereitgestellt, Betriebssysteme installiert, Softwaredateien kopiert, Komponenten konfiguriert sowie abschließend jeder einzelne Dienst manuell gestartet, sofern eine dienstorientierte Architektur zugrunde liegt [HF10]. Administratoren führen diese Schritte typischerweise ad-hoc auf Basis ihrer Erfahrung durch [EME+06; HdCG+14] oder folgen Dokumentationen wie „Best Practices“ [BH05], die beispielsweise durch die IT Infrastructure Library (ITIL) [Can07] oder durch Managementmuster [FLR+13; FLR+14] dokumentiert sind. In solchen manuellen Managementprozessen sind typischerweise eine große Anzahl verschiedener Managementaufgaben durchzuführen, welche großen Aufwand erfordern und insbesondere beim Management verteilter Anwendungen durch menschliche Umsetzungsfehler schnell zu fehlerhaften Konfigurationen, Datenverlusten oder dem Ausfall der gesamten Anwendung führen können. Erwiesenermaßen sind menschliche Fehler bei der Bereitstellung und dem Management komplexer IT-Systeme ein signifikanter Faktor für Systembeeinträchtigungen und Ausfälle [BP01; OGP03; Gad14; Fei15].

Dies resultiert zum einen aus der immensen Komplexität moderner IT-Systeme, die aus einer Vielzahl verteilter heterogener Dienste und Softwarekomponenten bestehen, welche konfiguriert und integriert werden müssen [STI+06; EME+06; HEW+09]. Insbesondere Cloud-basierte Anwendungen erfordern häufig die Kombination mehrerer Anbieter (engl. „Multi-Cloud Applications“), um Kosten zu senken oder die Verfügbarkeit zu erhöhen [FLRS12; SSSL12; FRC+13; Pet14]. Daraus resultieren komplexe Abhängigkeiten zwischen den einzelnen Bestandteilen einer Anwendung, wodurch sich lokale Änderungen häufig global auf andere Komponenten auswirken [DJ07]. Dem Administrator unbekannte Abhängigkeiten resultieren dabei schnell in Managementfehlern [BBK+14b]. Um eine Anwendung beispielsweise ohne Ausfallzeit in eine Cloud-Umgebung zu migrieren, müssen verschiedenste Paradigmen und Eigenschaften der involvierten Systeme berücksichtigt werden, um deren durchgängige Verfügbarkeit zu wahren [BBKL14c]. Umfangreiche Managementaufgaben, die mehrere Komponenten betreffen, erfordern somit enormes (i) *konzeptionelles Wissen*, wodurch die Komplexität des Managements steigt [BKH05; KBH07]. Andererseits erfordert das Management komplexer Systeme häufig die Kombination mehrerer Managementtechnologien, wozu Administratoren detaillierte (ii) *technische Expertise* über die eingesetzten Werkzeuge und Paradigmen aufweisen müssen [STI+06; BBK+13a; WBB15]. Soll z. B. eine Multi-Cloud-Anwendung provisioniert werden, müssen die proprietären und heterogenen Schnittstellen (APIs) der Cloud-Anbieter miteinander kombiniert und eventuell zusätzliche Technologien, wie Konfigurationsmanagementsysteme, integriert werden. Dazu müssen sowohl die Konzepte jeder Technologie verstanden sowie deren technische Bedienung bewältigt werden. Die Geschwindigkeit und Qualität der Ausführung hängt folglich direkt von der Qualifikation der beteiligten Personen ab, wodurch die Planungssicherheit beeinträchtigt wird [HF10]. Die manuelle Durchführung fordert somit großen Aufwand und Expertise über involvierte Komponenten, deren Abhängigkeiten und Managementtechnologien [EME+06], garantiert jedoch in keinsten Weise eine effiziente, planbare und fehlerfreie Ausführung. Daher ist der manuelle Ansatz vorwiegend zum Management überschaubarer Anwendungen geeignet, nicht jedoch zum Management komplexer Geschäftsanwendungen [STI+06].

Diese Probleme werden verstärkt, wenn keine ausreichend detaillierte Dokumentation der auszuführenden Schritte vorliegt, was aufgrund unterschiedlicher Wissensstände und Erfahrungen unterschiedlichen Personals innerhalb eines Unternehmens kaum in effizienter Form möglich ist [HF10]. Tritt ein Fehler auf, ist dessen Ursache häufig nicht offensichtlich, da manuelle Durchführungen aufgrund des immensen Aufwands nicht ausreichend dokumentiert werden und die tatsächlich durchgeführten Schritte somit im Detail schwer nachzuvollziehen sind. Zudem weichen menschliche Handlungen voneinander ab, wodurch die Reproduzierbarkeit und Korrektheit einer manuellen Managementdurchführung generell nicht gewährleistet werden kann. Dies beeinflusst unter anderem die Testphase, da nicht garantiert ist, dass Anwendungen in der Produktivumgebung identisch zur Testumgebung bereitgestellt werden [HF10].

Die manuelle Durchführung von Managementaufgaben erschwert zudem die Kollaboration mehrerer Beteiligter, da enormer Kommunikations- und Abstimmungsaufwand betrieben werden muss, um Konflikte zu vermeiden [HF10]. Müssen beispielsweise Managementaufgaben für eine Vielzahl verschiedener Anwendungsinstanzen ausgeführt werden, z. B. das Einspielen neuer Sicherheitsupdates, erfordert die manuelle Durchführung immensen Aufwand, dessen Erfolg direkt von der Koordination der involvierten Mitarbeiter abhängt. Da die wiederholte Ausführung solcher Prozesse für eine Vielzahl an Instanzen jedoch zu monotonen Abläufen führt, steigt die Wahrscheinlichkeit menschlicher Fehler signifikant an [HF10]. Dies führt zu erhöhtem Einarbeitungsbedarf in unterschiedlich konfigurierte Anwendungsinstanzen, wenn mehrere Mitarbeiter im Gesamtprozess involviert sind. Zudem führt manuelles Management generell zu Wissensverlusten, wenn Mitarbeiter das Unternehmen verlassen [HZR10].

Manuelles Management führt folglich zu (i) ineffizienten, (ii) zeit- und kostenintensiven sowie (iii) fehleranfälligen Abläufen. Diese Vorgehensweise ist somit nicht zum Management komplexer Geschäftsanwendungen geeignet. Daher beschreiben beispielsweise Brown und Keller [BK06] eine generelle Vorgehensweise zur systematischen Automatisierung manueller Unternehmensprozesse der IT-Dienstbereitstellung (engl. „*IT Service Delivery*“) und des Anwendungsmanagements auf Basis bewährten Wissens und ITIL Best Practices.

2.2.3.2 Automatisierte Managementdurchführung

Die Automatisierung des Anwendungsmanagements ist in vielen Fällen eine effektive Möglichkeit, um (i) manuelle menschliche Fehler bei der Managementdurchführung zu vermeiden, (ii) die Ausführung signifikant zu beschleunigen sowie (iii) die Kosten der Managementdurchführung zu reduzieren [FLRS12]. Automatisierung eliminiert somit eine Vielzahl der Probleme, die im Rahmen des manuellen Anwendungsmanagements auftreten. Insbesondere wenn ein aufwändiger Managementprozess häufig ausgeführt werden muss, etwa die Bereitstellung einer stark nachgefragten, komplexen Geschäftsanwendung, ermöglicht dessen Automatisierung eine wesentliche Zeitersparnis und Reproduzierbarkeit. Diese ist speziell für das Entwicklungsmodell der *Continuous Delivery* [HF10] und im Cloud Computing von grundlegender Bedeutung [FLRS12]. Durch die wiederholt identische Ausführung automatisierter Prozesse können diese effizient analysiert und optimiert werden, was bei manuellen Prozessen aufgrund häufiger Abweichungen deutlich aufwändiger ist. Zudem können automatisierte Prozesse überwacht und dokumentiert werden, wodurch die Nachvollziehbarkeit ausgeführter Schritte gewährleistet wird, z. B. zur Einhaltung der Firmen-Compliance oder von rechtlichen Rahmenbedingungen [WWB+13].

Zur Automatisierung des Anwendungsmanagements wurde in den vergangenen Jahren eine Vielzahl an Managementtechnologien entwickelt, welche von umfangreichen Managementdiensten der Cloud-Anbieter, über Modellierungssprachen für Managementprozesse, bis hin zu skriptbasierten Konfigurationsmanagementtechnologien kleiner Communities reichen. Eine detaillierte Übersicht der verschiedenen Technologien zur Automatisierung des Anwendungsmanagements wird in [Abschnitt 2.3](#) vorgestellt. Diese Technologien sind für überschaubare Anwendungsfälle und gewöhnliche Managementaufgaben effiziente Werkzeuge, um das Anwendungsmanagement zu automatisieren. Beispielsweise können moderne Cloud-basierte Plattformdienste die automatische Skalierung einer Anwendungskomponente übernehmen, ohne dass diese manuell durchgeführt, angestoßen oder überwacht werden muss [VRB11].

Bei der Automatisierung des Managements komplexer IT-Systeme treten jedoch ähnliche Probleme wie bei der manuellen Durchführung auf: (i) Zum einen wird konzeptionelle Erfahrung benötigt, um komplexe Managementaufgaben zu konzipieren. (ii) Zum anderen Bedarf es zum Management heterogener Anwendungskomponenten häufig verschiedene Managementtechnologien, welche folglich auch bei der Automatisierung auf technischer Ebene kombiniert und integriert werden müssen [BBK+13a]. Während dies auf manueller Ebene durch Expertise des durchführenden Systemadministrators bewerkstelligt wird, erfordert die Kombination verschiedener Managementtechnologien bei der Automatisierung des Gesamtprozesses eine technische Integration deren heterogenen Schnittstellen und Datenformate [BBK+13a]. Unterschiedliche Paradigmen, wie deklarative und imperative Ansätze, erschweren diese Integration fundamental [BBK+14a]. Um diese technischen Herausforderungen zu bewerkstelligen, ist somit technologiespezifische Expertise notwendig. Zudem muss eine Orchestrierungstechnologie eingesetzt werden, welche die Technologiaufrufe ausführt und den Datenfluss zwischen diesen koordiniert.

Diesen Herausforderungen stehen drei weitere Aspekte gegenüber, die gemäß Brown und Hellerstein [BH05] bei der Automatisierung von Managementprozessen beachtet werden müssen, da sie zu einer direkten Komplexitätssteigerung und zu versteckten Automatisierungskosten führen:

(i) Zur Nutzung einer Automatisierungstechnologie muss diese installiert und gewartet werden. Müssen für komplexe IT-Systeme mehrere Technologien kombiniert werden (vgl. [Abschnitt 2.2.3.1](#)), stellt dies einen erheblichen Aufwand dar, der vielseitige Expertise bezüglich deren Wartung erfordert.

(ii) Automatisierungstechnologien benötigen strukturierte Eingaben, die vorab konzipiert, erstellt und getestet werden müssen. Diese zusätzlichen, fixen Kosten beeinflussen signifikant die Höhe der Gesamtkosten, da viele Prozesse eine limitierte Lebenszeit haben, um diese Vorabaufwände wieder einzuspielen.

(iii) Die Erkennung und Behebung von Fehlern eines automatisierten Prozesses ist wesentlich komplexer als für manuelle Prozesse. Eine Ursache liegt in der höheren Flexibilität der manuellen Managementdurchführung: Scheitert die

Ausführung eines Managementschrittes, kann der Systemadministrator ad-hoc reagieren, um diesen zu korrigieren. In automatisierten Prozessen müssen solche Maßnahmen explizit implementiert werden, was angesichts der enormen Menge möglicher Fehler eine komplexe und aufwändige Aufgabe darstellt.

Wenn diese zusätzlichen Kosten die Vorteile der Automatisierung überwiegen, tritt die als *Ironie der Automatisierung* (engl. „*Irony of Automation*“ [Bai83]) beschriebene Situation ein, dass Automatisierung, die eigentlich zur Senkung der Kosten beabsichtigt war, ironischerweise die Gesamtkosten erhöht [BH05]. Automatisierung kann die Kosten somit negativ beeinflussen, wenn sie nicht unter ganzheitlicher Beachtung der Gesamtprozesse durchgeführt wird [BH05]. Es muss daher individuell abgewogen werden, wann sich diese zusätzlichen Kosten und die erhöhte Komplexität gegenüber der manuellen Managementdurchführung rentieren. In vielen Szenarien ist die Automatisierung jedoch nicht diskutabel, z. B. im Cloud Computing. Daher spielt die Optimierung der Automatisierung eine grundlegende Rolle im modernen Anwendungsmanagement.

2.2.4 Diskussion und offene Forschungsfragen

Die Komplexität des Anwendungsmanagements resultiert gemäß der vorangegangenen Analyse zum einen aus (i) konzeptionellen Herausforderungen, zum anderen aus (ii) technischen Schwierigkeiten, welche sich im Wesentlichen aus der nötigen Integration verschiedener Managementtechnologien ergeben. Die Umsetzung und Integration der beiden Abstraktionsebenen spielt in dieser Arbeit eine grundlegende Rolle, da diese in vollautomatisierter Form bisher nicht von existierenden Ansätzen ermöglicht wird. Die Reduktion der verschiedenen Komplexitätsdimensionen ist dabei von fundamentaler Bedeutung, um Managementautomatisierung effizient zu gestalten. Aufgrund der hohen Komplexität moderner IT-Systeme und deren Managementtechnologien muss die Kooperation unterschiedlicher Experten ermöglicht werden [EKK+06], ohne dass diese durch hohe Kommunikationsaufwände belastet werden oder in Zuständigkeitskonflikten enden. Die klassische Trennung der Verantwortlichkeiten (engl. „*separation of concerns*“) hat daher einen bedeutenden Einfluss

auf das Anwendungsmanagement, da dieses aufgrund der enormen Menge an benötigter Expertise bezüglich verschiedener Konzepte und Technologien nicht mehr von einzelnen durchgeführt werden kann. Die Konzeption der Durchführung einer Managementaufgabe kann daher in vielen Fällen grundsätzlich unabhängig von deren Ausführung durchgeführt werden. Beispielsweise kann ein Manager auf Architekturebene aufgrund neuer Geschäftsanforderungen Entscheidungen treffen ohne sich über die technische Ausführung Gedanken zu machen [dCMA+09]. Eine weitere Herausforderung der Integration resultiert aus den erforderlichen Interventionsmöglichkeiten durch den zuständigen Systemadministrator, da dieser für die Managementdurchführung verantwortlich ist und ein manueller Eingriff somit jederzeit möglich sein muss. Ein monolithisches Managementsystem, welches die Managementdurchführung vollständig übernimmt, jedoch keine manuelle Adaption und Individualisierung der Prozesse zulässt, ist somit nicht geeignet. Zudem muss das System robust und verlässlich sein, um auch das Management von komplexen Geschäftsanwendungen zu ermöglichen, die aufgrund ihrer Größe, Struktur oder ihrer Bedeutung für ein Unternehmen keine reaktive Fehlerbehandlung zulassen.

In der vorliegenden Arbeit werden die oben diskutierten Probleme sowie die drei versteckten Kostenfaktoren der Managementautomatisierung explizit angegangen. Mit der PALMA-Methode wird ein Vorgehensmodell vorgestellt, das die Managementdurchführung sowohl auf (i) konzeptioneller als auch auf (ii) technischer Ebene unterstützt und automatisierbar macht. Dabei wird konzeptionelles Wissen in Form von automatisierten Managementmustern anwendbar und die technische Integration sowie die Orchestrierung verschiedener Managementtechnologien durch die Wiederverwendung lauffähiger Managementbausteine automatisiert. Dadurch wird sowohl die Wartung als auch die Erstellung benötigter Eingaben jeder Managementtechnologie gekapselt, sodass die ersten beiden versteckten Kostenfaktoren eliminiert werden. Durch das PALMA-Framework wird die Anzahl zu bedienender Managementtechnologien auf eine reduziert, wodurch der erste Kostenfaktor bestmöglich eingegrenzt wird. Die Behandlung auftretender Fehler wird durch automatisierte Mechanismen unterstützt, wodurch der dritte Kostenfaktor eingedämmt wird.

2.3 Managementparadigmen und -technologien

In diesem Kapitel werden unterschiedliche Managementparadigmen vorgestellt und der Stand der Wissenschaft und Technik bezüglich existierender Managementtechnologien diskutiert. Um die verschiedenen Ansätze gegeneinander abgrenzen und vergleichen zu können, wird in [Abschnitt 2.3.1](#) zunächst eine Klassifikation von Managementaufgaben vorgestellt, um die grundlegenden Unterscheidungsmerkmale verschiedener Technologien zu charakterisieren. Da Managementtechnologien häufig ihre Schnittstellen in Form von APIs zur Verfügung stellen, müssen diese gegebenenfalls von anderen Managementtechnologien aufgerufen werden. Daher werden Management-APIs und deren Interaktionsmöglichkeiten in [Abschnitt 2.3.2](#) vorgestellt und diskutiert. [Abschnitt 2.3.3](#) stellt deklarative Managementkonzepte vor, die bisher vorwiegend bei der Bereitstellung von Anwendungen und deren Konfigurationsmanagement eingesetzt werden. Gleichmaßen diskutiert [Abschnitt 2.3.4](#) imperative Ansätze, die zur Beschreibung und Automatisierung von Managementprozessen eingesetzt werden, und evaluiert deren Vor- und Nachteile. Aufgrund der unterschiedlichen Stärken wurden zudem hybride Ansätze entwickelt, um die jeweiligen Vorteile zu vereinen. Diese werden in [Abschnitt 2.3.6](#) vorgestellt. Abschließend evaluiert [Abschnitt 2.3.8](#) die vorgestellten Paradigmen generell hinsichtlich deren Eignung zum Management komplexer Anwendungen.

2.3.1 Klassifikation von Managementaufgaben

Managementaufgaben werden in dieser Arbeit in zwei disjunkte Klassen unterteilt: (i) *zustandsmodifizierende* und (ii) *zustandserhaltende Managementaufgaben*. Aufgaben der ersten Klasse beeinflussen demnach den Zustand einer Anwendung, worunter auch Änderungen an deren Struktur und Konfiguration gesehen werden. Soll beispielsweise eine virtuelle Maschine provisioniert werden, unterscheidet sich die ursprüngliche Struktur der Anwendung von derjenigen nach Ausführung der Managementaufgabe dahingehend, dass die neue Komponente der Anwendungsstruktur hinzugefügt wurde. Zustandsmodifizierende Managementaufgaben können daher durch Beschreibung des *Zielzustands*

spezifiziert werden. Im Umkehrschluss können die zum Erreichen dieses Zielzustands auszuführenden Managementoperationen durch Vergleich von Ist- und Zielzustand abgeleitet werden. Diese Eigenschaft des Aufgabentyps legt die Grundlage vieler deklarativer Managementansätze (siehe [Abschnitt 2.3.3](#)). Zustandsmodifizierende Managementaufgaben werden wie folgt definiert:

Definition 4 (Zustandsmodifizierende Managementaufgaben - informell). Zustandsmodifizierende Managementaufgaben verändern den Zustand einer Anwendung. Dabei können sowohl die Anwendungsstruktur betroffen sein als auch die Konfigurationen und Zustände der Komponenten und Relationen. ■

Im Gegensatz dazu können *zustandserhaltende* Managementaufgaben nicht durch Spezifikation des gewünschten Zielzustands beschrieben werden, da diese weder den Zustand, die Konfiguration noch die Struktur der Anwendung beeinflussen. Sollen z. B. Daten aus einer Datenbank exportiert werden, ändert sich weder die Struktur der Anwendung noch der Zustand der Datenbank.

Definition 5 (Zustandserhaltende Managementaufgaben - informell). Zustandserhaltende Managementaufgaben verändern den Zustand einer Anwendung nicht. Sie beeinflussen weder die Struktur der Anwendung noch die Zustände und Konfigurationen deren Komponenten und Relationen. ■

Um beliebige Managementaufgaben modellieren und durchführen zu können, muss eine Technologie beide Typen unterstützen. Die Unterscheidung dieser Klassen spielt eine grundlegende Rolle bei der folgenden Evaluation unterschiedlicher Managementparadigmen, da imperative Ansätze generell beide Typen unterstützen, deklarative Technologien jedoch zumeist auf zustandsmodifizierende Aufgaben limitiert sind. Die in dieser Arbeit vorgestellte PALMA-Methode und die deklarative Modellierungssprache DMMN unterstützen beide Klassen von Managementaufgaben und können somit zur Modellierung und Ausführung umfangreicher Managementaufgaben eingesetzt werden.

2.3.2 Management-APIs

Viele Managementtechnologien stellen Programmierschnittstellen (engl. „*Application Programming Interfaces*“ (APIs) [RAR13]) zum Aufruf deren Funktionalität zur Verfügung, beispielsweise Cloud-Managementsysteme wie OpenStack¹ oder Cloud-Anbieter wie Amazon². Da Geschäftsanwendungen immer mehr auf verteilten Infrastrukturen betrieben werden, müssen unterschiedliche APIs sowohl von deklarativen als auch von imperativen Ansätzen eingebunden werden. Daher werden APIs vor der Evaluation der Managementparadigmen diskutiert.

Management-APIs basieren typischerweise auf dem HTTP-Protokoll und können daher über standardisierte Kommunikationswege aufgerufen werden, wodurch eine lose Kopplung bezüglich deren Implementierung ermöglicht wird. Viele Managementtechnologien können ausschließlich über APIs gesteuert werden, wodurch zum Management komplexer Geschäftsanwendungen häufig mehrere APIs orchestriert werden müssen [BBK+13a]. Ein grundlegendes Problem dieser APIs ist jedoch deren Heterogenität zueinander, die zu einer starken Kopplung an Datenformate und Operationsparameter führt: Obwohl immer mehr APIs auf HTTP als Transportprotokoll und REST [Fie00] als Architekturstil basieren, sind diese in den meisten Belangen weder syntaktisch noch semantisch interoperabel [SM11; DBC+12; BBK+13a; Pet14]. Zudem verändern sich APIs typischerweise während ihrer Lebensdauer [WKZ14], wodurch es zu fehlerhaften Aufrufen bzw. ungewollten Effekten kommen kann. Typische API-Änderungen werden von Wang et al. [WKZ14] analysiert und klassifiziert.

Ein Ansatz zur Stabilisierung von APIs und zur Reduktion deren Heterogenität stellen Standards dar. Das *Open Cloud Computing Interface (OCCI)* [OGF09] der OGF und das *Cloud Infrastructure Management Interface (CIMI)* [DMT12] der DMTF sind ähnliche Standards, um Managementschnittstellen standardisiert zu beschreiben [PZF13]. OCCI definiert eine REST / HTTP-basierte Schnittstelle und API-Spezifikation, um verschiedene Typen an Cloud-Ressourcen zu verwalten und auszuführende Managementaufgaben zu spezifizieren, beispielsweise

¹<http://developer.openstack.org/>

²<http://docs.aws.amazon.com/AWSEC2/latest/APIReference>

um eine virtuelle Maschine zu stoppen. CIMI definiert ebenfalls eine REST / HTTP-basierte Schnittstelle, um Managementinteraktionen zwischen einem Cloud-Infrastrukturanbieter und dem Nutzer zu spezifizieren. Im Gegensatz zu OCCl, welches Cloud-Ressourcen auf verschiedenen Ebenen wie IaaS, PaaS und SaaS beschreiben kann, fokussiert CIMI die Infrastrukturschicht [DMT12]. Die *Cloud Application Management for Platforms*-Spezifikation (CAMP) [OAS12] ist ein OASIS Standard, der eine generische API-Beschreibung sowie ein Paketformat definiert, um Anwendungen auf Plattformdiensten (PaaS) bereitzustellen, zu überwachen und zu managen. Die Standards sind komplementär zu dieser Arbeit und erleichtern die Entwicklung wiederverwendbarer Managementlogik.

2.3.3 Deklarative Managementansätze

In diesem Abschnitt werden Sprachen und Technologien des *deklarativen Managementparadigmas* vorgestellt. Um die Semantik des Begriffs der „Deklarativität“ eindeutig für das Anwendungsmanagement zu definieren, wird zunächst dessen Verwendung in anderen IT-Bereichen untersucht. In der Domäne der Prozessmodellierung unterscheidet Pesic [Pes08] folgendermaßen zwischen *prozeduraler Modellierung*, welche in der vorliegenden Arbeit mit *imperativer Modellierung* gleichzusetzen ist, und *deklarativer Modellierung*: Imperative Prozessmodelle folgen einem „*inside-to-outside*“ Ansatz, wodurch alle möglichen Ausführungsalternativen der Aktivitäten eines Prozesses explizit spezifiziert sind. Im Gegensatz dazu sind deklarative Prozessmodelle invers zu diesem Ansatz und beschreiben Prozesse „*outside-to-inside*“, indem Einschränkungen (engl. „*constraints*“) implizit spezifizieren, welche Ausführungsalternativen der modellierten Aktivitäten möglich sind. In ähnlicher Form existieren im Bereich der Datenbanken *deklarative Abfragesprachen*, mithilfe derer Einschränkungen an die gewünschten Daten spezifiziert werden können, und *prozedurale Abfragesprachen*, welche die explizite Spezifikation von Sequenzen auszuführender Operationen ermöglichen [EN10]. Lloyd [Llo94] definiert Deklarativität in der Programmierung dadurch, dass bei dieser angegeben wird, *was* berechnet werden soll, jedoch nicht zwingendermaßen, *wie* eine Berechnung durchzuführen ist. Äquivalent zur Terminologie von Kowalskis Gleichung „*algorithm = logic + control*“ [Kow79] wird bei der deklarativen Programmierung somit die Lo-

gik eines Algorithmus angegeben [Llo94], aber nicht die zu deren Umsetzung auszuführenden Anweisungen. Basierend auf diesen Interpretationen wird der Begriff folgendermaßen für das Anwendungsmanagement definiert:

Definition 6 (Deklaratives Anwendungsmanagement - informell). Deklaratives Anwendungsmanagement bezeichnet die Durchführung von Managementaufgaben auf Basis einer domänenspezifischen deklarativen Spezifikation, die Managementziele vorgibt, welche von einer Ausführungsumgebung umgesetzt werden. Deklarative Spezifikationen geben an *was* erreicht werden soll, jedoch ohne die Anweisungen zu beschreiben, *wie* die Ziele zu erreichen sind. ■

2.3.3.1 Deklarative Konfigurationsmanagementwerkzeuge

Managementaufgaben bezüglich des Konfigurationsmanagements von Anwendungen werden häufig durch imperative Shell-Skripte automatisiert, welche jedoch mit zunehmender Anwendungsgröße eine erhöhte Komplexität erreichen und zudem an die unterliegende Ausführungsumgebung gekoppelt sind. Um die Komplexität von Shell-Skripten zu reduzieren und das Problem der schlechten Portierbarkeit dieser Skripte zu lösen, entwickelte Mark Burgess 1993 das Systemadministrationswerkzeug *CFEngine* [Bur95]. CFEngine basiert auf einer deklarativen domänenspezifischen Sprache, mithilfe derer oft gebrauchte Programmierparadigmen zur Konfiguration von Unix-Systemen durch einfache deklarative Vorschriften ersetzt werden können.

Eine Vielzahl an Werkzeugen, die dem *DevOps-Paradigma* [Hüt12] folgen, ermöglichen die Automatisierung der Installation und Konfiguration von Software. Konfigurationsmanagementwerkzeuge wie *Ansible* [MR14], *Chef* [Ops16; TV14], *Puppet* [Pup16; Uph14] oder *Salt* [Sal16] setzen dazu ebenso deklarative domänenspezifische Sprachen zur Spezifikation von Zielkonfigurationen ein, welche automatisiert auf mehreren physikalischen und virtualisierten Maschinen zur Laufzeit durch Ausführung technischer Anweisungen hergestellt werden. Durch deklarative DSLs wird so eine Abstraktionshöhe geschaffen, welche die Spezifikation gewünschter Konfigurationen unabhängig von der letztendlichen Zielumgebung ermöglicht, da diese erst zur Laufzeit für die

jeweilige Plattform in konkrete technische Anweisungen übersetzt werden. Die von vielen Werkzeugen zusätzlich ermöglichte imperative Spezifikation auszuführender technischer Anweisungen schränkt diese Portabilität jedoch ein. Die genannten Werkzeuge fokussieren dabei stark die Konfiguration von Komponenten auf Ebene der unterliegenden Maschinen, wodurch deren Kombination mit anderen Technologien, beispielsweise Management APIs, eine signifikante Herausforderung darstellt, falls diese durch vollautomatisierte, integrierte Prozesse ausgeführt werden sollen. Zudem fokussieren diese Technologien ausschließlich die Spezifikation gewünschter Konfigurationen von Komponenten, wodurch zustandserhaltende Managementaufgaben nicht bzw. nur eingeschränkt modelliert werden können. In der vorliegenden Arbeit werden diese Art Werkzeuge bei der Implementierung von Management-Planlets eingesetzt, um Betriebssysteme und Softwareinstallationen zu konfigurieren.

Um den traditionell manuellen Entscheidungsprozess zur Konfiguration eines Systems zu unterstützen und zu automatisieren, stellen Hewson et al. mit *ConfSolve* [HA11; HAG12] eine objektorientierte deklarative Sprache zur Modellierung von Systemkonfigurationen vor, die es erlaubt, Einschränkungen (engl. „constraints“) über das Zielsystem zu spezifizieren, z. B. dass zwei Anwendungskomponenten nicht auf demselben physikalischen Server betrieben werden dürfen. Durch Formulierung eines *ConfSolve*-Modells als *Constraint Satisfaction Problem (CSP)* können CSP-Solver dazu verwendet werden, automatisiert (i) die Korrektheit der modellierten Konfiguration zu verifizieren, (ii) Einflüsse der Konfiguration zu analysieren und (iii) valide Konfigurationen zu generieren bzw. (iv) diese zu optimieren. In der vorliegenden Arbeit werden Einschränkungen in dieser Form nicht behandelt, stehen jedoch komplementär zu den vorgestellten Ansätzen und sind Teil zukünftiger Forschungsarbeiten.

2.3.3.2 Zielzustandsmodelle

In den vorangegangenen Abschnitten wurden zahlreiche Sprachen zur Beschreibung von Anwendungen und deren Konfiguration vorgestellt, welche zu deren automatisierten Bereitstellung und Konfiguration eingesetzt werden. Diese reichen von graphbasierten Topologiemodellen, welche die Beschreibung der Anwendungsstruktur ermöglichen, bis hin zu weniger strukturierten

Spezifikationen einzelner Komponentenkonfigurationen, die von deklarativen Konfigurationsmanagementwerkzeugen umgesetzt werden. Eine gemeinsame Eigenschaft all dieser Modellarten liegt darin begründet, dass diese zur deklarativen, präskriptiven Beschreibung eines *Ziels* verwendet werden - sei es die gewünschte Struktur einer bereitzustellenden Anwendung oder die Neukonfiguration einer bereits betriebenen Komponente. Im Wesentlichen spezifizieren all diese Modelle daher einen bestimmten *Zustand*, der erreicht werden soll. Die in [Abschnitt 2.1.4](#) vorgestellten Sprachen zur Modellierung der Anwendungsbereitstellung sowie die Sprachen der in [Abschnitt 2.3.3.1](#) vorgestellten Konfigurationsmanagementwerkzeuge ermöglichen somit die Erstellung von Modellen, die generell unter dem Oberbegriff des *Zielzustandsmodells* zusammengefasst werden können. Um im Folgenden von irrelevanten Unterschieden bezüglich der Formalisierung diesbezüglicher Informationen zu abstrahieren, wird der Begriff wie folgt informell definiert:

Definition 7 (Zielzustandsmodell - informell). Ein Zielzustandsmodell beschreibt (i) implizit oder explizit die gewünschte Struktur, in die eine Anwendung oder Teile davon überführt werden sollen, sowie (ii) den individuellen Zustand deren Komponenten und Relationen, welchen diese einnehmen sollen. Unter dem Begriff *Zustand* werden sowohl Konfigurationen als auch beliebige Laufzeitinformationen über die jeweilige Komponente bzw. Relation zusammengefasst, beispielsweise auch Namen oder Prozessorauslastungen. ■

Zielzustandsmodelle sind in ihrer Natur deklarativ und weitgehend als geeignetster Ansatz zur Anwendungsbereitstellung und zum Konfigurationsmanagement akzeptiert [[HAW11](#)]. Dieser Definition entsprechende Modelle werden in zahlreichen wissenschaftlichen Forschungsarbeiten dazu eingesetzt, Konzepte, Methoden und Algorithmen zur Umsetzung der modellierten Ziele zu entwickeln. Diese werden im Folgenden vorgestellt und diskutiert. Die Arbeiten werden dabei in Ansätze zur direkten Operationsausführung ([Abschnitt 2.3.3.3](#)) und Ansätze zur Generierung von Workflow-Modellen ([Abschnitt 2.3.3.4](#)) unterteilt. Zielzustandsmodelle werden in diesen Ansätzen im Englischen typischerweise als „*Desired State Model*“ bezeichnet.

2.3.3.3 Ansätze zur direkten Operationsausführung

Zur Umsetzung von Zielzustandsmodellen berechnen die folgenden Ansätze (i) direkt auszuführende Anweisungen oder (ii) delegieren die Umsetzung gewünschter Zustandsübergänge an dedizierte Managementkomponenten.

Morin et al. [MBJ+09] stellen einen modellgetriebenen Ansatz zur dynamischen Adaption von Systemen (engl. „*Dynamically Adaptive Systems*“) vor, der das Instanzmodell einer Anwendung verwendet, um auf Basis dessen eine neue Konfiguration der Anwendung zu beschreiben. Diese wird anschließend durch Vergleich von Ist- und Zielkonfiguration zur Ableitung auszuführender Anweisungen herangezogen. Das System unterstützt jedoch nur Managementaufgaben, die den Zustand bzw. die Struktur der Anwendung verändern, indem Komponenten und Relationen erzeugt bzw. entfernt werden. Dolstra et al. [DVL13] stellen mit *Charon* ein Provisionierungs- und Konfigurationswerkzeug vor, das deklarative Zielzustandsmodelle dazu einsetzt, um virtuelle Maschinen zu provisionieren und mittels des Paketmanagers *Nix* [DvJ04] zu konfigurieren sowie diese miteinander über ein Netzwerk zu verbinden. Das *SmartFrog* [GGL+09] Konfigurationsmanagementsystem definiert eine deklarative Sprache in Form eines Komponentenmodells zur Modellierung von Anwendungskonfigurationen. Dabei kann jede Komponentendeklaration mit einem sogenannten *Lifecycle Manager* assoziiert werden, der für die Interpretation und Umsetzung der jeweiligen Komponentenkonfiguration zuständig ist. Papaioannou et al. [PMM15] erweitern *SmartFrog* um die Fähigkeit, virtuelle Maschinen unterschiedlicher Typen dynamisch in privaten und öffentlichen Cloud-Infrastrukturen zu provisionieren. Auf Basis einer Metadatenbank, welche aktuelle Zustands- und Strukturinformationen laufender Anwendungen enthält, können so beispielsweise Migrationsaufgaben durchgeführt werden, indem äquivalente Maschinen auf einer neuen Infrastruktur provisioniert werden. Einen anderen Ansatz verfolgen Herden et al. [HZR10], die eine Systemarchitektur vorstellen, welche die automatisierte Bereitstellung und Konfiguration von Anwendungskomponenten auf Basis von *Deployment Daemons* und *Wenn-Dann-Regeln* (engl. „*Deklarative Rule Pattern*“) ermöglicht. Diese übersetzen deklarativ spezifizierte, zustandsbezogene Managementziele in imperative Anweisungen.

2.3.3.4 Ansätze zur Generierung von Workflow-Modellen

In diesem Abschnitt werden Arbeiten vorgestellt, die explizite Workflow-Modelle generieren, deren Ausführung eine Anwendung aus dem Ist- in einen modellierten Zielzustand überführen. Zunächst werden Ansätze vorgestellt, die das Problem der Workflow-Generierung auf *AI-Planning-Probleme* (engl. „*Artificial Intelligence Problem*“) reduzieren [EME+06]. Da sich die Arbeiten in der konzeptionellen Technik ähnlich sind, wird diese zunächst abstrahiert zusammengefasst. Anschließend werden unterschiedliche Umsetzungen diskutiert.

Zur Generierung von Workflow-Modellen konsumieren die im Folgenden vorgestellten AI-Planning-Algorithmen¹ (i) das Instanzmodell der Anwendung, (ii) das Zielzustandsmodell sowie (iii) die Menge aller verfügbaren atomaren Managementoperationen, die ausgeführt werden können, um eine Komponente oder Relation von einem Zustand in einen anderen zu überführen. Diese Managementoperationen werden als *Aktionen* bezeichnet und im Rahmen einer *Planning-Domäne* zusammengefasst. Jede dieser Aktionen kann dabei nur ausgeführt werden, wenn deren formal erfasste *Vorbedingungen* (engl. „*pre-conditions*“) bezüglich Komponenten- und Relationenzuständen erfüllt sind. Beispielsweise, dass die Komponente, auf welcher eine Aktion Software installieren soll, bereits läuft. Die Änderungen, die eine Aktion am Zustand der Anwendung durchführt, werden dabei ebenfalls formal in Form von *Effekten* (engl. „*effects*“) spezifiziert. Auf Basis dieser Spezifikationen können AI-Planner die zum Erreichen des Zielzustands nötigen Aktionen sowie deren kausale und temporale Abhängigkeiten berechnen, aus denen anschließend ein ausführbares Workflow-Modell generiert wird, welches die Aktionsimplementierungen orchestriert. Diese Berechnung wird typischerweise in Form von Suchalgorithmen umgesetzt, die verschiedene Kombinationen von Aktionen solange untersuchen, bis eine vollständige und konfliktfreie Lösung gefunden wird [EME+06]. Während der Berechnung basiert die Auswahl und Ordnung geeigneter Aktionen naturgemäß auf nicht-deterministischen Entscheidungen, die mittels Backtracking in deterministische Anweisungen übersetzt werden [Wei94].

¹Im Englischen häufig nur als *AI-Planner* bezeichnet.

Eine generische Sprache zur Beschreibung von AI-Planning-Problemen stellt die *Planning Domain Definition Language (PDDL)* [MGH+98] dar, die über das Anwendungsmanagement hinaus in verschiedensten Domänen als Industriestandard eingesetzt wird [AHS05]. Die zu erreichenden Ziele des Planning-Problems sowie Vorbedingungen von Aktionen werden in PDDL in funktionsfreier Prädikatenlogik erster Stufe (engl. „*function-free first-order predicate logic*“) beschrieben [MGH+98]. Angewendet auf das Anwendungsmanagement ermöglicht PDDL die Formalisierung von Instanzmodellen, Zielzustandsmodellen sowie von Aktionen in einer einheitlichen Syntax, wodurch keine diesbezüglichen Modelltransformationen während der Berechnung der Lösung, d. h. während der Generierung des Workflow-Modells, erforderlich sind.

Arshad et al. [AHW03; AHW07] zeigen erstmalig, wie Zielzustandsmodelle, PDDL und AI-Planning zur Generierung von Workflow-Modellen eingesetzt werden können, um die Konfiguration laufender, verteilter Anwendungen zu modifizieren (engl. „*dynamic reconfiguration*“). Insbesondere beschreiben die Autoren, wie während der Workflow-Ausführung aufgetretene Fehler durch eine automatische Generierung eines neuen Zielzustandsmodells und *Replanning* behandelt werden können, um die Anwendung in einen konsistenten, korrekten Anwendungszustand zu führen. Andrzejak et al. [AHS05] beschreiben ein Framework, das PDDL dazu verwendet, zustandsmodifizierende Managementaktionen, Instanzmodelle sowie Zielzustandsmodelle deklarativ zu beschreiben und automatisiert durch den *MBP-Planner* [BCP+01] in Workflow-Modelle zu transformieren. Managementaktionen werden dabei durch Java-Klassen implementiert, welche die Ausführung von Ant-Skripten via SSH auf dem Zielsystem steuern. Maghraoui et al. [EME+06] greifen diesen Ansatz auf und wenden ihn auf die Bereitstellung von Anwendungen an. Die Autoren stellen vor, wie Instanzmodelle und Zielzustandsmodelle in Prädikatenlogik erster Stufe formalisiert werden können, um diese durch *Partial Order Planning (POP)* [Wel94] in ausführbare, parallelisierte Workflow-Modelle zu transformieren. Aktionen werden dazu ebenfalls in PDDL beschrieben. Zudem stellen die Autoren eine Systemarchitektur vor, um die auszuführenden Aktionen über Adapter mit Bereitstellungssystemen zu verbinden, und diskutieren Optimierungsmöglich-

keiten hinsichtlich des Planungsvorgangs auf Basis genereller Charakteristika der Domäne. Partial Order Planning wird in der vorliegenden Arbeit ebenfalls zur Generierung ausführbarer Workflow-Modelle eingesetzt, um die parallele Ausführung unabhängiger Managementaufgaben zu ermöglichen und dadurch die Ausführungszeit des Workflows zu optimieren.

Die Generierung von Workflow-Modellen ist in den bisher vorgestellten Ansätzen nicht durch den Administrator beeinflussbar oder einschränkbar. In einigen Managementszenarien, beispielsweise beim Umzug eines Dienstes auf einen anderen Server, spielt jedoch die Ausführungsreihenfolge der Operationen eine signifikante Rolle - etwa wenn andere Komponenten, welche den umzuziehenden Dienst nutzen, keinen Ausfall erfahren sollen. In diesem Beispiel muss der Dienst zuerst neu bereitgestellt und die nutzenden Komponenten über den aktualisierten Endpunkt informiert werden, bevor der alte Dienst außer Betrieb genommen wird. Da diese Art der *Einschränkungen* (engl. „constraints“) in vielen Ansätzen nicht spezifizierbar ist bzw. manuell in generierten Workflow-Modellen umgesetzt werden muss, stellen Herry et al. [HAW11] einen AI-Planning-basierten Ansatz vor, um Workflows zur Überführung einer Anwendung aus dem aktuellen Istzustand in einen deklarativ modellierten Zielzustand unter Einhaltung spezifizierter Einschränkungen zu generieren. PDDL wird dabei zur Beschreibung von Zustandsmodellen und Aktionen verwendet, ControlTier und Puppet zur Realisierung der Workflows. Einschränkungen werden direkt in den Vorbedingungen der Aktionen spezifiziert. Beispielsweise spezifiziert die Aktion zur Deinstallation des Dienstes die Einschränkung, dass es keine darauf zugreifenden Klienten geben darf. Zur Modellierung diesbezüglicher Einschränkungen müssen somit die Vorbedingungen der Aktionen adaptiert werden, was die Wiederverwendbarkeit der Aktionen in anderen Szenarien einschränkt [HA12]. Sind die nutzenden Komponenten des Szenarios beispielsweise so implementiert, dass diese vom umzuziehenden Dienst nicht direkt abhängig sind und diesen nur optional nutzen, wird die Workflow-Generierung unter Umständen durch unnötig eingeschränkte Aktionen negativ beeinflusst. In einer späteren Arbeit [HA12] lösen die Autoren dieses Problem, indem die globale Spezifikation von Einschränkungen direkt im Zielzustandsmo-

dell ermöglicht wird. In der vorliegenden Arbeit werden daher Einschränkungen nicht auf Ebene der Aktionen, sondern ebenfalls auf Ebene des deklarativen Zielmodells spezifiziert. In einer späteren Arbeit [HAR13] erweitern die Autoren diese Ansätze zudem, um die generierten Workflow-Modelle verteilt durch Agenten in einer Choreographie auszuführen. Die verteilte Ausführung der generierten Modelle ist jedoch nicht Gegenstand der vorliegenden Arbeit.

Die Modellierung des Zielzustands erfordert in den bisher vorgestellten Ansätzen eine detaillierte Kenntnis über die involvierten Systeme: Komplexe Kombinationen aus Konfigurationsparametern verschiedener Komponenten müssen so gewählt werden, dass gewünschte Eigenschaften und Anforderungen aus der Geschäftsperspektive, beispielsweise eine hohe Verfügbarkeit des Gesamtsystems, erfüllt werden. Dies erschwert die Modellierung von Zielzustandsmodellen und resultiert häufig in ungeeigneten Konfigurationen. Zur Behebung dieser Probleme stellen Levanti und Ranganathan [LR09] einen AI-Planning-basierten Ansatz vor, der es dem Administrator ermöglicht, die gewünschte (i) Zielkonfiguration sowie (ii) High-Level Eigenschaften der Anwendung in Form von deklarativen, abstrakten *Tags* zu modellieren. Zum Beispiel, dass ein System gestartet werden soll und dass dieses eine hohe Verfügbarkeit gewährleisten muss. Diese *Tags* abstrahieren von technischen Konfigurationsparametern und unterstützen dadurch unerfahrene Administratoren bei der Spezifikation gewünschter Ziele. Das Zielzustandsmodell wird somit durch eine Menge an *Tags* modelliert die zu erfüllen sind, wodurch technische Konfigurationsparameter nicht spezifiziert werden müssen. Zudem werden *Tags* eingesetzt, um Eingaben, Ausgaben, Vorbedingungen sowie Effekte von Aktionen zu beschreiben. Um *Tags* auf feingranulare, technische Konfigurationsparameter abzubilden, die Aktionen als Eingabe benötigen, werden zudem *Optionsmodule* eingesetzt, die wie Aktionen aufgebaut sind, allerdings keine Änderungen an der Anwendung durchführen und lediglich abstrakte *Tags* in technische Konfigurationsparameter übersetzen. Basierend auf diesem Tag-basierten Metamodell wird AI-Planning eingesetzt, um ein Workflow-Modell aus Aktionen und Optionsmodulen zu generieren, welche die spezifizierten *Tags* umsetzen. Die vorliegende Arbeit reduziert diese Komplexität dadurch, dass Managementziele auf hoher deklarativer Abstrakti-

onsebene direkt an den betroffenen Komponenten und Relationen deklariert werden können, die anschließend von Management-Planlets umgesetzt werden. Ein wesentlicher Unterschied zum Ansatz von Levanti und Ranganathan ist dabei, dass die vorliegende Arbeit auch zustandserhaltende Managementaufgaben sowie die Modellierung deren Abhängigkeiten unterstützt.

Neben diesen Ansätzen, welche allesamt Zustandstransitionen durch extern spezifizierte Operationen beschreiben und durchführen, werden im Folgenden Komponentenmodelle und Sprachen vorgestellt, die Transitionen in Form von Zustandsautomaten direkt an der betroffenen Komponente modellieren.

Hagen und Kemper [HK10a] stellen einen Planning-Ansatz vor, der auf einem objektorientierten Komponentenmodell basiert, in welchem jede Komponente den aktuellen Zustand, z. B. *Installed*, und Laufzeitinformationen getrennt beschreibt. Jede Komponente spezifiziert zudem ein Zustandstransitionssystem (engl. „*State Transition System*“) sowie Operationen, die mit Vorbedingungen und Effekten bezüglich der Laufzeitinformationen und mit Zustandsübergängen im Transitionssystem assoziiert sind. Durch Aufruf von Operationen kann eine Komponente somit in einen anderen (formal spezifizieren) Zustand überführt werden. Durch AI-Planning wird über eine Tiefensuche eine Sequenz möglicher Operationsaufrufe gesucht, welche die Anwendung von ihrem Ursprungszustand in einen modellierten Zielzustand überführen. Zur Optimierung des Algorithmus werden Techniken untersucht, um Zwischenzustände während der Planung im Falle des Backtrackings effizient wiederherzustellen.

Das Komponentenmodell *Aeolus* [DZZ12; DMZZ14] greift dieses Konzept auf und erweitert es um die Modellierung von Anwendungsstrukturen in Form von typisierten, zustandsbehafteten Komponenten, die (i) angebotene Funktionalitäten (*Provided Ports*) sowie (ii) Funktionalitäten, welche die Komponente von anderen Komponenten benötigt (*Required Ports*), an interne Managementzustände koppelt. Jede Komponente spezifiziert interne Zustandstransitionen dabei durch einen Zustandsautomaten (engl. „*State Machine*“), wobei zum Übergang von einem Zustand in einen anderen Zustand möglicherweise Funktionalitäten von anderen Komponenten benötigt bzw. eigene Funktionalitäten diesen durch den neuen Zustand zur Verfügung gestellt werden. *Aeolus* ermöglicht zudem die

Beschreibung des gewünschten Zielzustandes einer Anwendung, wozu Lascu et al. [LMZ13; LMZ15] einen Planning-basierten Ansatz vorstellen, um einen Deployment-Plan zu generieren, welcher die Anwendung in den beschriebenen Zustand überführt. Auf Basis der formalen Grundlagen von Aeolus stellen Di Cosmo et al. [DLT+14] weitere Werkzeuge vor, die eine abstrakte Beschreibung eines gewünschten Systems und die dazu verfügbaren Komponententypen als Eingabe erhalten und daraus ein feingranulares Architekturmodell generieren, das anschließend auf virtuellen Maschinen mittels Paketen installiert werden kann. In ähnlicher Form führen Kuroda und Gokhale [KG14b] ein Komponentenmodell ein, das Transitionen zwischen Zuständen von Komponenten mit Operationen assoziiert und basierend darauf die Ableitung auszuführender Operationen durch Vergleich von Ist- und Zielzustandsmodell ermöglicht. Dadurch können mehrere Zwischenzustände einer Komponente durchlaufen werden bis der modellierte Zielzustand erreicht ist. In ähnlicher Form wird der Ansatz von Kuroda und Gokhale [KG14a] auch eingesetzt, um auszuführende Aufgaben zur Provisionierung physischer Hardwarekomponenten zu berechnen.

Die bisher vorgestellten Techniken basieren auf AI-Planning, das jedoch aufgrund inhärenter Komplexität die Formalisierung von Aktionen erschwert. Insbesondere stellen Eilam et al. [EEKS11] fest, dass die Beschreibung der Planning-Domäne, d. h. die Modellierung von Aktionen, mittels PDDL eine Herausforderung für Modellierer darstellt, die objektorientierte Modelle gewohnt sind. Um diese Komplexität zu reduzieren, formalisieren die Autoren in derselben Arbeit die Funktionalität von Aktionen in Form von Topologiefragmenten¹. Die darin enthaltenen Komponenten spezifizieren Vorbedingungen zur Ausführung der Aktion sowie deren Effekte in Form von Attributen. Zudem kann sowohl der Ausgangszustand bezüglich des Lebenszyklus der Komponente, z. B. *Installed* oder *Running*, als Vorbedingung sowie deren Zielzustand als Effekt der Aktion spezifiziert werden. Dasselbe Metamodell wird zur Modellierung von zustandsmodifizierenden Managementaufgaben in Form topologiebasierter Zielzustandsmodelle eingesetzt. Die Eignung einer Aktion zum Erreichen

¹Diese Topologiefragmente sind kleine Anwendungstopologien, die in der Publikation als „*Model-based Patterns*“ bezeichnet werden.

eines Zielzustands einzelner oder mehrerer Komponenten wird dabei auf ein Subgraphisomorphismusproblem reduziert: Kann das Topologiefragment der Aktion unter Erfüllung aller Vorbedingungen auf einen Subgraphen des Zielzustandsmodells abgebildet werden und entsprechen die Effekte der Aktion den spezifizierten Zielen, dann kann die Aktion zur Erreichung des modellierten Zielzustands der abgebildeten Komponenten eingesetzt werden. Die Auswahl aller nötigen Aktionen zur Generierung des Workflows wird auf ein Mengenüberdeckungsproblem (engl. „*set cover problem*“) [GJ79] reduziert: Für jeden spezifizierten Zielzustand muss eine Aktion gewählt werden, welche diesen erzielt. Die Ausführungsreihenfolge, d. h. die partielle Ordnung, ausgewählter Aktionen wird durch Vergleich von Effekten und Vorbedingungen berechnet. Durch diese Technik kann der Workflow für jede Komponente genau eine Aktion ausführen, um den spezifizierten Zielzustand zu erreichen. Dadurch sind mehrere Zwischenzustände nicht möglich. Während diese Einschränkung für die Bereitstellung von Anwendungen akzeptabel ist [EEKS11], müssen zur Umsetzung umfangreicher Managementaufgaben häufig mehrere Aktionen bezüglich einzelner Komponenten und Relationen ausgeführt werden. Beispielsweise wenn eine Komponente getestet werden soll und dazu vorübergehend Ports geöffnet werden müssen, die anschließend wieder zu schließen sind. In der vorliegenden Arbeit wird die topologiebasierte Form der Beschreibung von zustandsbezogenen Vorbedingungen und Effekten einer Aktion mit AI-Planning kombiniert, wodurch Vorteile beider Ansätze vereint werden: (i) Die Beschreibung der Funktionalität von Management-Planlets, die in der Terminologie des AI-Planning Aktionen darstellen, basiert auf einem intuitiven objektorientierten Metamodell. Zudem (ii) erlaubt DMMN die Spezifikation komplexer Managementziele, die den Zustand einer Komponente unter Umständen mehrfach modifizieren. Eine grundlegende Erweiterung stellt des Weiteren die Unterstützung von zustandserhaltenden Managementaufgaben dar, wodurch auszuführende Managementaufgaben explizit deklariert und nicht nur über Zustandstransitionen beschrieben werden können.

Während die bisher vorgestellten deklarativen Ansätze eine explizite Spezifikation der gewünschten Anwendungsstrukturen und Zielzustände erfordern, stellen Delaet und Joosen mit *PoDIM* [DJ07] eine objektorientierte Sprache vor, mithilfe derer gewünschte Konfigurationen deklarativ auf Basis von Regeln und Einschränkungen modelliert werden können. *PoDIM* definiert dazu ein Komponentenmodell, das Zustandstransitionen mit Operationen einer Komponente assoziiert, sowie eine Regelsprache, um die gewünschte Konfiguration zu spezifizieren. Diese ermöglicht sowohl die Deklaration der Bereitstellung gewünschter Komponenten als auch diesbezüglicher Einschränkungen, beispielsweise, dass vier virtuelle Maschinen bereitgestellt werden sollen, von denen zwei als Webserver und zwei als Datenbanken agieren sollen. Die tatsächliche Konfiguration wird anschließend von einem *Controller* unter Einhaltung aller spezifizierten Regeln berechnet und ausgeführt.

2.3.3.5 Diskussion: Einschränkungen der vorgestellten deklarativen Ansätze

Die deklarative Modellierung von Managementaufgaben durch Spezifikation der zu erreichenden Zielzustände bietet ein einfach erlernbares, intuitives Managementparadigma zur Beschreibung zustandsmodifizierender Managementaufgaben: Es müssen weder technisch komplexe Anweisungen noch deren Kontroll- und Datenfluss modelliert bzw. implementiert werden, wodurch eine Abstraktion hinsichtlich der tatsächlichen Managementdurchführung geschaffen wird. Speziell die Ansätze von Levanti und Ranganathan [LR09] unterstützen dies durch Abstraktion feingranularer technischer Zustände zu höherwertigen abstrakten Tags. Einige Ansätze kombinieren den deklarativen Ansatz zudem mit dem imperativen Ansatz, um ausführbare Workflow-Modelle zu generieren. Dadurch werden die Stärken beider Paradigmen vereint, indem deklarative Zielbeschreibungen eine einfache Beschreibung der Ziele ermöglichen, wohingegen die generierten Workflow-Modelle beliebig vom Administrator modifiziert und bearbeitet werden können. In der vorliegenden Arbeit wird daher zur Ausführung deklarativer DMMN-Modelle ebenfalls ein Generierungsverfahren eingesetzt, das auf AI-Planning basiert und die Workflow-Technologie zur Ausführung der modellierten Managementaufgaben einsetzt.

Eine fundamentale Einschränkung der in diesem Kapitel vorgestellten Arbeiten resultiert aus der fehlenden Unterstützung zustandserhaltender Managementaufgaben. Die existierenden deklarativen Ansätze ermöglichen zwar die Spezifikation zustandsmodifizierender Managementaufgaben, indem der gewünschte Zielzustand modelliert wird, nicht jedoch zustandserhaltende Managementaufgaben, die den Zustand der Anwendung nicht beeinflussen. Dieser Aufgabentyp kann weder durch Spezifikation einer strukturell modifizierten Anwendungsstruktur noch durch die Spezifikation bestimmter Zielzustände modelliert werden. Daher sind zustandserhaltende Managementaufgaben, z. B. der Export von Daten aus einer Datenbank, nicht direkt deklarativ modellierbar, was die Hauptlimitation dieser Konzepte, Sprachen und Technologien darstellt.

Eine Lösung dieses Problems bieten Ansätze, die Workflow-Modelle zur Erreichung der deklarierten Ziele generieren. Die generierten Workflow-Modelle können beliebig vor deren Ausführung modifiziert werden, um beispielsweise manuell zustandserhaltende Managementaufgaben hinzuzufügen. Dies erfordert jedoch in großem Maße manuellen Aufwand und führt aufgrund der hohen Komplexität zu einer fehleranfälligen Herausforderung. Insbesondere wird im folgenden Kapitel im Rahmen der Diskussion imperativer Managementansätze festgestellt, dass die manuelle Adaption imperativer Workflow-Modelle, speziell das Hinzufügen neuer Managementaufgaben, nicht effizient durchführbar ist. Daher sollte diese Vorgehensweise nur in Sonderfällen herangezogen werden.

Eine weitere Limitation deklarativer Ansätze ist die eingeschränkte Integrationsmöglichkeit externer Technologien zur Ausführung anwendungsspezifischer Managementaufgaben. Obwohl Aktionen prinzipiell beliebig implementiert werden können, wirkt sich die Modifikation der Planning-Domäne auf das Management aller Anwendungen aus und ist somit zur Behandlung anwendungsspezifischer Anforderungen nicht geeignet. Insbesondere stellt die korrekte Modellierung und Formalisierung von Aktionen eine komplexe Herausforderung dar, wodurch sich dieses Vorgehen auch nicht zur temporären Modifikation der Domäne eignet. Einen Ansatz zur Lösung dieses Problems ermöglichen Technologien, die einen Komponentenlebenszyklus definieren, dessen Phasen individuell implementiert werden können und deren Aufruf durch das Manage-

mentssystem in einer vordefinierten Reihenfolge erfolgt. Da die Ausführungsreihenfolge dieser Standardphasen strikt vorgegeben ist, kann der Administrator jedoch nur innerhalb dieser Phasen die Ausführung beeinflussen. Der Gesamtprozess kann somit bezüglich anwendungsspezifischer Anforderungen nicht beliebig deklarativ beeinflusst werden, wodurch die Individualisierbarkeit stark eingeschränkt ist. In der vorliegenden Arbeit werden diese Probleme gelöst, indem Managementaufgaben zum einen deklarativ sowohl modelliert als auch, falls erforderlich, temporal hinsichtlich deren Ausführungsreihenfolge geordnet werden können. Dadurch kann die Ausführung der deklarativen Managementbeschreibung gezielt gesteuert und eingeschränkt werden, wodurch imperative Charakteristika ermöglicht und die Managementdurchführung beliebig vom zuständigen Administrator beeinflusst werden kann. Zum anderen ermöglicht das Konzept der Management-Planlets die nahtlose Integration externer Technologien zur Ausführung der deklarierten Managementaufgaben, wodurch diese nicht nur im Rahmen von Standardphasen integriert werden können, sondern an beliebigen Stellen des Managementprozesses individuell einsetzbar sind.

Zielzustandsmodelle unterliegen aufgrund der Singularität modellierter Zielzustandswerte einer weiteren Limitation, indem ausschließlich finale Endzustände spezifiziert werden können, nicht jedoch zu durchlaufende Zwischenzustände. Dies führt bei Managementaufgaben, die aufgrund besonderer Anforderungen einzelner Anwendungen speziell und unterschiedlich zu deren typischen Umsetzung durchgeführt werden müssen, zu aufwändigen Individualbehandlungen. Wird zum Beispiel das Einspielen eines Webserver-Patches im Regelfall im Produktivbetrieb und ohne besondere Vorkehrungen durchgeführt, kann es hinsichtlich einer wichtigen Geschäftsanwendung erforderlich sein, dass diese währenddessen auf einen anderen Webserver migriert wird, um das Ausfallrisiko zu minimieren. Um diese Art individueller Behandlung umzusetzen, erfordern die meisten vorgestellten Ansätze die Modifikation der Planning-Domäne, d. h. die Modifikation der Vorbedingungen verfügbarer Aktionen, oder die manuelle Anpassung des generierten Workflow-Modells. Während der Ansatz von Herry et al. [HAW11] die Spezifikation einzuhaltender Rahmenbedingungen ermöglicht, um diese Art Probleme zu lösen, ermöglicht DMMN die explizite Modellierung

mehrerer Zwischenzustände, die durchlaufen werden müssen. Dadurch können Individualbehandlungen explizit deklarativ modelliert werden, während die standardmäßige Ausführung der Managementaufgaben mit derselben Planning-Domäne realisiert werden kann. Des Weiteren können in DMMN beide Klassen an Managementaufgaben temporal geordnet werden, was speziell zu deren Kombination häufig erforderlich ist. Insbesondere scheitert das Konzept der Zielzustandsmodelle, falls zudem zustandserhaltende Managementaufgaben benötigt werden, beispielsweise, wenn Daten vor Einspielung des Patches als Backup exportiert werden sollen. In DMMN-Modellen können Zielzustände somit systematisch deklarativ mit zustandserhaltenden Managementaufgaben kombiniert werden.

Einige proprietäre Ansätze sind an bestimmte Anbieter oder Technologien gekoppelt und eignen sich daher nicht zum Management beliebiger Anwendungen. Deklarative Sprachen großer Cloud-Anbieter, wie beispielsweise *Cloud-Formation* [Ama16] von Amazon, eignen sich daher nur für Anwendungen, die auf den jeweiligen Infrastrukturen bzw. Plattformen der Anbieter betrieben werden. Da Anwendungskomponenten, z. B. im Rahmen von Multi-Cloud-Anwendungen [Pet14], aufgrund unterschiedlicher Anforderungen bezüglich Performance, Kosten und Funktionalität zunehmend auf mehrere, heterogene Infrastrukturen und Plattformen verteilt werden, erfordert die Umsetzung *holistischer Managementprozesse* eine Kombination verschiedener Technologien. Da dies von proprietären Sprachen und Technologien nicht nativ unterstützt wird, eignen sich diese Ansätze nicht zur Bereitstellung und Verwaltung beliebiger Anwendungen. Dies stellt eine Grundherausforderung der vorliegenden Arbeit dar und wird durch die generische Managementmodellierungssprache DMMN und dem Konzept der Management-Planlets explizit unterstützt.

2.3.4 Imperative Managementansätze

In diesem Abschnitt werden Konzepte, Sprachen und Technologien des *imperativen Managementparadigmas* vorgestellt. Imperative Ansätze charakterisieren sich dadurch, dass sie auszuführende Anweisungen im Detail beschreiben und explizit spezifizieren (i) *was* ausgeführt werden soll und (ii) *wie* die Ausführung erfolgen soll. Durch dieses Paradigma können beliebige Managementaufgaben beschrieben, konfiguriert und automatisiert ausgeführt werden. Das imperative Anwendungsmanagement wird wie folgt informell definiert:

Definition 8 (Imperatives Anwendungsmanagement - informell). Imperatives Anwendungsmanagement bezeichnet die Durchführung von Managementaufgaben auf Basis einer domänenspezifischen imperativen Spezifikation, die explizit alle auszuführenden Anweisungen sowie deren Reihenfolge festlegt. ■

Durch die Möglichkeit beliebige Anweisungen explizit beschreiben zu können, unterstützen imperative Sprachen grundsätzlich sowohl die Beschreibung zustandsmodifizierender als auch zustandserhaltender Managementaufgaben. In der vorliegenden Arbeit wird zwischen drei imperativen Managementansätzen unterschieden, welche die Beschreibung auszuführender Anweisungen auf unterschiedliche Arten ermöglichen: (i) Programme, (ii) Skripte und (iii) Workflows [LR00]. Diese werden im Folgenden vorgestellt und hinsichtlich ihrer Eignung zur Umsetzung der PALMA-Methode diskutiert.

2.3.4.1 Managementprogramme

Aufgrund des langjährigen Wachstums von Informationstechnologie in Unternehmen wurden zur Automatisierung von Managementprozessen ähnliche Technologien eingesetzt wie zur Implementierung von Anwendungen. Aus diesem Grund werden viele Managementprozesse in Unternehmen durch monolithische Managementprogramme realisiert, welche in Programmiersprachen wie Java implementiert sind. Entwickler und Administratoren konnten durch diese imperativen Beschreibungen frühzeitig Managementaufgaben automatisieren, die aufgrund domänenspezifischer Technologien der damaligen Zeit zu

Altlasten führten, welche heute sukzessive durch neue Technologien abgelöst werden. Ein wesentlicher Nachteil dieser Automatisierungsweise liegt in der limitierten Flexibilität und Anpassungsfähigkeit: Monolithische Managementprogramme bilden Managementlogik direkt in Programmcode ab, wodurch die Integration anderer Systeme sowie die offene Kooperation behindert werden [BZ95]. In der vorliegenden Arbeit spielen monolithische Programme, die Managementprozesse fest verdrahtet implementieren (engl. „*hard coded*“) um diese zu automatisieren, keine Rolle hinsichtlich der Orchestrierung verschiedener Technologien, da moderne Ansätze wie die Workflow-Technologie aufgrund der signifikant höheren Flexibilität und Robustheit deutlich besser für ein effizientes Anwendungsmanagement geeignet sind (siehe [Abschnitt 2.3.4.3](#)). Das vorgestellte Konzept zur automatisierten Generierung von Management-Workflow-Modellen unterstützt daher explizit die Integration von Altlasten in Form solcher „*Legacy Management Software*“ (siehe [Abschnitt 5.6](#)).

2.3.4.2 Managementskripte

Managementskripte sind Programme, die eine Sequenz feingranularer Anweisungen spezifizieren [EKK+06; WBL14a]. Sie werden vorwiegend dazu eingesetzt, um Software zu installieren, zu konfigurieren und zu deinstallieren. Da Skripte typischerweise interpretiert werden, sind sie an spezielle Ausführungsumgebungen gebunden und werden meistens auf Betriebssystemen ausgeführt, um lokale Installationen oder Änderungen an installierter Software durchzuführen [EKK+06]. Sie eignen sich somit vorwiegend dazu, auf physikalischen oder virtuellen Maschinen Softwareänderungen durchzuführen. Bekannte Skriptsprachen sind beispielsweise Perl, Python und Unix Shell-Skripte. Diese imperativen Sprachen wurden von Systemadministratoren primär genutzt bevor Konfigurationsmanagementwerkzeuge, wie beispielsweise Puppet [Pup16], und deren domänenspezifische Sprachen entwickelt wurden, die nach und nach Einzug in modernes Konfigurationsmanagement nehmen [WBL14a] (vgl. [Abschnitt 2.3.3.1](#)). Managementskripte werden typischerweise manuell erstellt [HAW11], was aufgrund der niedrigen Abstraktionshöhe eine komplexe und fehleranfällige Aufgabe darstellt [EKK+06].

Während sich Skripte im Konfigurationsmanagement durchgesetzt haben, eignet sich diese Technologie aufgrund ihrer Unzuverlässigkeit [HAW11] nicht zur Implementierung umfangreicher Managementprozesse, deren Umsetzung die Orchestrierung einer Vielzahl an unterschiedlichen Managementtechnologien erfordert. Zudem unterstützten viele Ausführungsumgebungen für Skripte keine Recovery-Mechanismen, sodass die idempotente Ausführung von Skripten eine fundamentale Implementierungsanforderung darstellt: Treten Fehler auf oder fällt die Ausführungsumgebung aus, beispielsweise durch einen Stromausfall, wird ein idempotentes Skript solange wiederholt ausgeführt, bis es fehlerfrei terminiert. Dabei können jedoch unerwünschte Seiteneffekte auftreten, beispielsweise wenn ein Deinstallationsskript eine Komponente bei der ersten Ausführung nicht vollständig deinstalliert und die Komponente dadurch bis zu einer späteren, erfolgreichen Skriptausführung ein fehlerhaftes Verhalten aufweist. Werden jedoch externe APIs aufgerufen, um beispielsweise eine virtuelle Maschine bereitzustellen, erfordert diese Vorgehensweise eine besondere Implementierung, da das Skript nicht einfach wiederholt werden kann ohne die ursprüngliche Provisionierung abzubrechen oder die bereits bereitgestellte virtuelle Maschine bei der erneuten Ausführung des Skripts wiederzuverwenden. Aufgrund der Komplexität des Anwendungsmanagements ist dieses Vorgehen für anwendungsspezifische Komponenten nicht effizient umsetzbar, da eine Vielzahl an individuellen Sonderfällen und Ausnahmebehandlungen implementiert werden müssen, um die erforderliche Idempotenz herzustellen.

Limitierte Fehlerbehandlungs- und Kompensationsmechanismen stellen zudem eine große Hürde dar, um Anwendungsmanagement allein durch Skripte umzusetzen. Wettinger et al. [WBL14a; WBL15b] behandeln dieses Problem durch die Einführung von *Kompensationsskripten*. Zur Orchestrierung verschiedener, heterogener Managementtechnologien eignet sich jedoch die robuste Workflow-Technologie, die im nächsten Abschnitt 2.3.4.3 vorgestellt wird, deutlich besser.

2.3.4.3 Management-Workflows

Die Workflow-Technologie ermöglicht die Automatisierung von Geschäftsprozessen durch Modellierung deren Aktivitäten und Abläufe. Durch *Workflow-Modelle* können Geschäftsprozesse formal beschrieben und automatisiert mithilfe von *Workflow-Maschinen* (engl. „*workflow engines*“ [LR00]) ausgeführt werden. Eine Instanz eines Workflow-Modells wird als *Workflow* bezeichnet [LR00].

In den vergangenen Jahren haben sich zwei Standards in dieser Domäne durchgesetzt: (i) Die *Business Process Execution Language (BPEL)* [OAS07b] und (ii) die *Business Process Model and Notation (BPMN 2.0)* [OMG11b]. Diese ermöglichen die standardisierte Beschreibung auszuführender Geschäftsprozesse durch Modellierung von Aktivitäten, deren Ausführungsreihenfolge durch einen expliziten Kontrollfluss spezifiziert wird. Die Aktivitäten können dabei Abhängigkeiten bezüglich Eingabe- und Ausgabedaten haben, deren Fluss implizit bzw. explizit durch das Workflow-Modell beschrieben werden kann [KKL08].

Workflow-Maschinen ermöglichen eine robuste Ausführung der modellierten Prozesse [LR00; Ley10]. *Recovery-Mechanismen* gewährleisten die nahtlose Fortsetzung aktiver Instanzen, falls Störungen zu einem Absturz der Workflow-Maschine führen, beispielsweise im Falle eines Stromausfalls. Durch diese als *Phoenix-Verhalten* (engl. „*Phoenix Behaviour*“) bezeichnete Eigenschaft sichern Workflow-Maschinen die Integrität und Konsistenz des Workflows und der involvierten Daten [LR00]. Weitere Sprachkonzepte zur Fehlerbehandlung und Kompensation unterstützen dieses Verhalten und ermöglichen eine verlässliche, nachvollziehbare und reproduzierbare Ausführung. Angesichts dieser Eigenschaften wird die Workflow-Technologie bereits von einigen Arbeiten zur Bereitstellung [KB04; MUL09; Mie10; BCFP13; BBK+14a] und zum Management [KHW+04; KBBL12; BBKL13b; BBH+13] von Anwendungen eingesetzt.

Aufgrund der Robustheit dieser Technologie werden Workflows in der vorliegenden Arbeit zur Automatisierung der PALMA-Methode eingesetzt: Durch die Musteranwendung erzeugte DMMN-Modelle werden im Rahmen von [Forschungsbeitrag 3](#) in ausführbare Workflow-Modelle transformiert und durch eine Workflow-Maschine ausgeführt. Dadurch wird gewährleistet, dass die Aus-

führung unbeeinflusst von auftretenden Störungen vollständig durchgeführt oder durch Einsatz von Fehlerbehandlungs- und Kompensationsmechanismen die Konsistenz des Anwendungszustandes gewahrt wird. Im Vergleich zu Programmen oder Skripten stellt die Workflow-Technologie somit eine verlässliche Grundlage zur Orchestrierung von Managementtechnologien dar. In der vorliegenden Arbeit werden Workflow-Modelle, die zur Ausführung von Managementaufgaben eingesetzt werden, wie vom TOSCA-Standard definiert [OAS13b] als *Managementpläne* bezeichnet (Synonym: *Management-Workflow-Modelle*).

Zur hierarchischen Strukturierung von Prozessen werden sogenannte *Subprozesse* verwendet. Diese stellen ein Modellierungskonstrukt dar, um Prozesse zu modularisieren, in separate Modelle auszulagern und wiederzuverwenden [LR00]. Ein *Vaterprozess* kann Subprozesse aufrufen, um deren Funktionalität auszuführen. Ein *traditioneller Subprozess* wird gemäß Kopp et al. [KELU10] dadurch charakterisiert, dass er syntaktisch genau einen Anfang sowie genau ein Ende hat (*Single-Entry-Single-Exit, SESE*) und darüber hinaus nicht mit dem aufrufenden Vaterprozess kommunizieren darf. Ein weiterer wesentlicher Gesichtspunkt eines traditionellen Subprozesses stellt dessen Autonomie bezüglich seiner Lebensdauer dar, die er aufgrund der Abhängigkeit zum aufrufenden Vaterprozess aufgibt: Terminiert oder kompensiert der Vaterprozess, wird dasselbe auch für einen aufgerufenen Subprozess durchgeführt [KELU10]. Die Umsetzung von Subprozessen wird jedoch nicht einheitlich von Workflow-Sprachen unterstützt. So erfordert beispielsweise BPEL die Erweiterung *BPEL-SPE* [KKL+05], um die Autonomie eigenständiger Subprozesse (engl. „*Standalone Sub-processes*“) aufzugeben, während das dynamische Konzept der *Worklets* [AtHEvA06] entwickelt wurde, um in der Workflow-Sprache YAWL [vAtH05] einen Subprozess in Abhängigkeit des Kontextes der Prozessinstanz auszuwählen. Ein Überblick unterschiedlicher Auslegungen wird von Kopp et al. [KELU10] gegeben.

Das Konzept der Aufteilung von Logik in Unterprozesse und deren Autonomieaufgabe wird in der vorliegenden Arbeit dazu verwendet, um Managementlogik, die durch ein Workflow-Modell implementiert ist, in Form von traditionellen Subprozessen in anderen Management-Workflows wiederzuverwenden. Dadurch können kleinere Managementaufgaben für bestimmte Komponenten

und Relationen unabhängig von individuellen Anwendungen in Form von ausführbaren Workflow-Modellen modularisiert beschrieben und zur Ausführung umfangreicherer Managementaufgaben orchestriert werden. Diese Subprozesse werden in der vorliegenden Arbeit als *Management-Planlets* bezeichnet.

Die Workflow-Standards BPEL und BPMN sind generische Sprachen zur Automatisierung von Geschäftsprozessen (engl. „*general-purpose workflow languages*“). Da die Domäne des Anwendungsmanagements eine Vielzahl heterogener Managementtechnologien umfasst, welche häufig miteinander kombiniert werden (vgl. [Abschnitt 2.2.3](#)), müssen Managementprozesse diese orchestrieren. Während Web-basierte APIs aufgrund des standardisierten Transportprotokolls durch native Konstrukte der Workflow-Sprachen eingebunden werden können, ist die Integration proprietärer Managementtechnologien deutlich komplexer und erfordert in vielen Fällen die Implementierung von Adaptern oder Erweiterungen, beispielsweise durch *BPEL Extension Activities* [[OAS07b](#)]. Um die Integration verschiedener Technologien zu unterstützen, wurde von Kopp et al. eine domänenspezifische Sprache namens BPMN4TOSCA [[KBBL12](#)] entwickelt, welche auf Basis des TOSCA-Standards die Modellierung abstrakter Aufrufe beliebiger Managementtechnologien ermöglicht. Mithilfe des von Wettinger et al. [[WBB+14b](#); [WBB+14a](#)] vorgestellten Managementbusses werden die abstrakt spezifizierten Aufrufe auf konkrete Managementtechnologien abgebildet, wodurch ein generisches Adapterkonzept für Managementtechnologien geschaffen wurde. In ähnlicher Form stellen Weerasiri et al. [[WBB15](#)] eine Spracherweiterung für Workflows vor, die einen neuen Aktivitätstyp für das Deployment von Cloud-Ressourcen definiert. Um diese Aktivität generisch zu definieren, wird eine abstrakte Schnittstellenbeschreibung eingeführt, welche Managementoperationen unabhängig von konkreten Anbietern und Technologien definiert. Diese Schnittstelle wird durch sogenannte *Resource Configuration Services* anbieterspezifisch implementiert, um verschiedene Technologien zu integrieren. Aufgrund der noch nicht abgeschlossenen Entwicklung von BPMN4TOSCA und des Deployment-Fokus der Ansätze von Weerasiri et al. werden diese DSLs in der vorliegenden Arbeit nicht zur Umsetzung von Management-Planlets eingesetzt, eignen sich aber perspektivisch zur Vereinfachung deren Entwicklung.

Cordeiro et al. [dCMD+08; dCMA+09] führen ein Metamodell ein (basierend auf CIM [DMT10]), um gewünschte IT-Änderungen in Form von sogenannten *Change Templates* zu spezifizieren. Dabei wird zwischen *Request Templates*, welche einen Request for Change (RFC) [Can07] auf hoher Abstraktionsebene beschreiben, und *Plan Templates* unterschieden, welche komponierbares Managementwissen in Form von abstrakten Workflows modellieren. Diese Plan Templates werden dazu verwendet, einen (auf Basis von Request Templates) spezifizierten RFC manuell in einen sogenannten *Change Plan* zu übersetzen, welcher die grundlegenden Schritte zur Ausführung der Managementaufgabe in abstrakter Form modelliert. Die manuell modellierten Change-Pläne werden anschließend automatisiert durch Verfeinerung der abstrakten Aktivitäten in ausführbare Workflow-Modelle übersetzt und ausgeführt [dCMA+08]. Durch Kombination beider Dokumentationsformen kann Managementwissen somit auf unterschiedlichen Ebenen spezifiziert und zudem wiederverwendbar als Vorlage für weitere Ausführungen dokumentiert werden. Die Ansätze werden durch das Managementsystem *ChangeLedge* implementiert [dCMA+09]. Die vorliegende Arbeit führt diese Motivation fort, indem abstraktes Managementwissen durch automatisierte Managementmuster dokumentiert und dadurch ohne manuelle Intervention auf Instanzmodelle laufender Anwendungen angewendet werden kann, um die gewünschten IT-Änderungen deklarativ in Form von DMMN-Modellen zu spezifizieren. Die resultierenden Modelle werden anschließend ebenfalls automatisiert in Workflow-Modelle übersetzt.

OpenTOSCA [BBH+13] ist ein Open-Source Managementsystem zur Bereitstellung und Verwaltung von Cloud-Anwendungen. Das System implementiert den TOSCA-Standard [OAS13b; OAS13a] (siehe Abschnitt 2.1.4) und stellt mit dem grafischen TOSCA-Modellierungswerkzeug *Winery* [KBBL13] und dem Selbstbedienungsportal *Vinothek* [BBKL14d] eine Werkzeugkette dar, um Cloud-Anwendungen (i) zu modellieren, (ii) bereitzustellen und (iii) zu managen. OpenTOSCA ermöglicht die Ausführung von Managementaufgaben durch Workflows, die in BPEL modelliert sind. Daher eignet sich die Laufzeitumgebung prinzipiell zur Ausführung von Managementplänen, die mittels der vorgestellten Konzepte der vorliegenden Arbeit generiert werden.

2.3.5 Diskussion - Implementierung imperativer Managementprozesse

Die vorgestellten imperativen Managementansätze, Programme, Skripte und Workflows, unterscheiden sich deutlich in (i) Granularität und technischer Ausrichtung, (ii) im Vorgehen der Implementierung sowie (iii) in deren Eignung zur verlässlichen Ausführung von Managementaufgaben. Während Programme und Skripte typischerweise technische Anweisungen auf tiefer Operationsebene implementieren, eignen sich Workflows auf höherer Abstraktionsebene, um diese Implementierungen und andere Managementtechnologien zu orchestrieren. In Anlehnung an die Unterscheidung „*Programming-In-The-Large*“ und „*Programming-In-The-Small*“ [DK75] werden die unterschiedlichen Managementebenen in der vorliegenden Arbeit als „*Application-Management-In-The-Large*“ und „*Application-Management-In-The-Small*“ bezeichnet, wobei deren Übergang in Abhängigkeit der jeweiligen Sprache bzw. Technologie fließend ist. Skripte, Programme, APIs und deklarative Managementtechnologien¹ sind daher „in-the-small“ angesiedelt, wohingegen sich die Workflow-Technologie zu deren Orchestrierung „in-the-large“ eignet.

Technologien, welche „in-the-small“ angesiedelt sind, werden meistens durch Anbieter bereitgestellt oder durch Communities unterstützt: APIs werden generell von Anbietern oder Komponenten selbst zur Verfügung gestellt und können damit direkt aufgerufen werden, während Konfigurationsmanagement-Communities eine Vielzahl wiederverwendbarer Artefakte anbieten, die zur Ausführung häufig wiederkehrender Managementaufgaben eingesetzt werden können (vgl. [Abschnitt 2.3.4.2](#)). Ähnlich zur Modularisierung von Anwendungskomponenten sind Artefakte und Schnittstellen der kleinen Kategorie somit nur an kleine Anwendungsstrukturen gekoppelt und daher in verschiedenen Kontexten wiederverwendbar: Ein Skript, das einen Webserver installiert, kann bei der Bereitstellung vieler umfangreicher Anwendungen eingesetzt werden, während dasselbe für den Aufruf eines API zur Provisionierung einer virtuellen

¹Die Ansiedlung deklarativer Managementtechnologien „in-the-small“ resultiert aus deren typischerweise engen Bindung an eine individuelle Umgebung, wie beispielsweise Cloud-Formation an Amazon gekoppelt ist, und deren limitierten Möglichkeiten zur Orchestrierung beliebiger Technologien (siehe [Abschnitt 2.3.3.5](#)). Workflows eignen sich zur Orchestrierung beliebiger Umgebungen, weswegen die Einteilung relativ im Vergleich mit Workflows zu sehen ist.

Maschine gilt. Im Gegensatz dazu stellt die Orchestrierung dieser Schnittstellen eine größere Herausforderung dar, da unterschiedliche domänenspezifische Sprachen, Datenformate, Protokolle und Schnittstellenbeschreibungen deren Integration signifikant erschweren und somit die Komplexität sowohl auf technischer als auch auf konzeptioneller Ebene erhöhen, wie bereits in [Abschnitt 2.2.3](#) diskutiert. Speziell die imperative Orchestrierung deklarativer Managementtechnologien stellt aufgrund der unterschiedlichen Paradigmen eine große technische Integrationshürde dar. Die manuelle Modellierung einer Orchestrierung durch ein Workflow-Modell ist daher komplex, fehleranfällig und aufwändig [[BBK+13a](#)]. Zudem sind Managementpläne, die eine umfangreiche Funktionalität modellieren, deutlich stärker an eine individuelle Anwendung gekoppelt und unterliegen den schon andiskutierten Alterungseffekten. Diese Probleme werden in den folgenden Abschnitten analysiert, da sie die Hauptmotivation der automatisierten Generierung von Managementplänen darstellen.

2.3.5.1 Kopplung von Managementplänen an individuelle Anwendungen

Managementpläne unterliegen generell einer engen Kopplung an individuelle Anwendungen. Diese resultiert aus zwei Aspekten: (i) Individuelle Anwendungsstrukturen sowie (ii) Heterogenität eingesetzter Managementtechnologien.

Managementpläne, die eine umfangreiche Managementfunktion wie die Migration von Komponenten zwischen verschiedenen Infrastrukturen modellieren, sind typischerweise von mehreren Komponenten und Relationen einer Anwendung abhängig [[EME+06](#)]. Soll beispielsweise ein Webserver in eine Cloud-Umgebung migriert werden, sind auch darauf betriebene Anwendungen betroffen und müssen gegebenenfalls gesondert betrachtet werden, insbesondere falls andere Komponenten auf diese zugreifen und nach der Migration aktualisierte Endpunkte benötigen. Um eine korrekte Ausführung der Managementaufgaben zu garantieren, muss daher die Struktur und Konfiguration der Anwendung bei der Modellierung von Managementplänen beachtet werden und der Plan explizit darauf abgestimmt werden: Der Anwendungszustand und deren Struktur, für die eine Managementaufgabe auszuführen ist, beeinflussen signifikant die Modellierung des Plans und dessen Ausführung [[BBK+14b](#)]. Dies führt zu

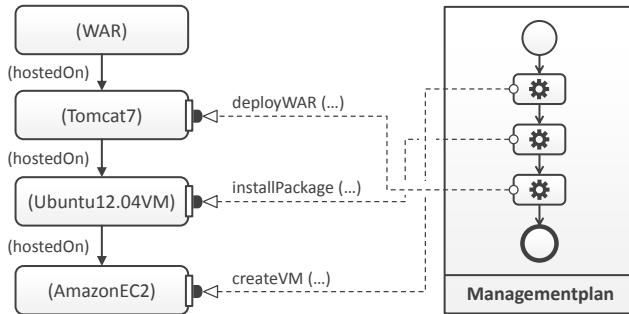


Abbildung 2.2: Kopplung von Managementplänen an Schnittstellen.

einer starken Kopplung von Plänen an individuelle Anwendungsstrukturen, deren Konfigurationen und mögliche Zustände, wodurch diese grundsätzlich individuell für eine Anwendung modelliert werden müssen [EME+06].

Eine weitere Kopplung resultiert aus der Diversität von Managementtechnologien, die aufgrund heterogener Managementschnittstellen nicht ohne Weiteres ausgetauscht werden können, da Managementpläne deren Aufrufe direkt spezifizieren. Durch Cloud Computing hat sich dieses Interoperabilitätsproblem zusätzlich verstärkt, da die Anzahl heterogener Managementschnittstellen rapide zugenommen hat: Modelliert ein Plan einen API-Aufruf eines auszutauschenden Cloud-Anbieters, kann dieser nicht einfach ersetzt werden, ohne den Aufruf an das API des neuen Anbieters anzupassen. Abstraktionstechnologien, wie z. B. *Deltacloud*¹, können dieses Problem reduzieren, im Allgemeinen jedoch nicht vollständig lösen, da nicht alle Anbieter dieselbe Funktionalität unterstützen.

Abbildung 2.2 veranschaulicht diese Kopplungsproblematik anhand eines Beispiels. Der Managementplan auf der rechten Seite implementiert die Bereitstellung einer Java-basierten Anwendung auf Amazons Infrastrukturdienst *Elastic Compute Cloud (EC2)*², welche als Anwendungstopologie auf der linken Seite dargestellt wird. Die Anwendungslogik ist in Form einer Java Web-Applikation im WAR-Format realisiert und soll auf einem Tomcat 7-Webserver³ betrieben

¹<https://deltacloud.apache.org/>

²<https://aws.amazon.com/ec2>

³<https://tomcat.apache.org>

werden, der auf einer virtuellen Maschine mit Ubuntu 12.04 Betriebssystem installiert werden soll. Dieser simple Managementplan ist jedoch direkt an drei Schnittstellen und deren Operationen gekoppelt. Wird beispielsweise ein anderer Cloud-Anbieter eingesetzt, der nur Windows-Betriebssysteme unterstützt, müssen zwei der drei Aktivitäten an die neuen Gegebenheiten angepasst werden: Weder bieten Cloud-Anbieter einheitliche Schnittstellen zur Provisionierung von virtuellen Maschinen an noch unterstützt Windows dasselbe Paketformat wie Linux. Der Plan ist somit eng an (i) die Anwendungsstruktur und (ii) die zur Bereitstellung benötigten Technologien gekoppelt.

2.3.5.2 Auswirkungen der Kopplung an individuelle Anwendungen

Die enge Kopplung von Managementplänen an individuelle Anwendungsstrukturen und Technologien resultiert in zwei negativen Eigenschaften: (i) Eingeschränkte Wiederverwendbarkeit und (ii) erhöhter Wartungsaufwand.

Zum einen reduziert die enge Kopplung den Grad der Wiederverwendbarkeit von Managementplänen drastisch und führt dazu, dass Managementaufgaben in dieser Größenordnung meist individuell für Anwendungen, unter Berücksichtigung deren Struktur und der involvierten Managementtechnologien, implementiert werden müssen [BBKL13b]. Dies resultiert darin, dass Expertenwissen sowohl auf konzeptioneller als auch auf technischer Ebene benötigt wird und dieses nur schwer auf andere Anwendungen übertragen werden kann, was letztendlich zu immensen Aufwänden für die Implementierung neuer Pläne führt. Zusätzlich ist speziell die technische Umsetzung bei einer manuellen Implementierung von Plänen, insbesondere die technische Integration heterogener Managementtechnologien, sehr zeitaufwändig und fehleranfällig [BBK+13a].

Zum anderen führen Änderungen an der Struktur, den Bestandteilen und der Konfiguration einer Anwendung, welche typischerweise nach längerem Betrieb nötig sind, zu zusätzlichem Wartungsaufwand bezüglich der zugehörigen Managementpläne. Wird beispielsweise ein Sicherheitsupdate für einen Webserver eingespielt, sind möglicherweise Managementoperationen, welche in der ursprünglichen Version noch verfügbar waren und verwendet wurden, nicht mehr

aufrufbar. Änderungen an APIs von Cloud-Anbietern und Managementtechnologien (siehe [Abschnitt 2.3.2](#)) müssen ebenso in Managementplänen berücksichtigt werden. Dies führt zu manuellen Anpassungen von Plänen im Laufe des Anwendungslebenszyklus, welche die oben beschriebenen Herausforderungen bezüglich der konzeptionellen als auch der technischen Implementierungskomplexität nach sich ziehen und zusätzlichen Aufwand verursachen.

2.3.5.3 Alterung von Managementprozessmodellen

In den vorangegangenen Kapiteln wurde in einigen Abschnitten eine gewisse *Alterung* imperativer Managementprozessmodelle festgestellt, welche direkt aus deren Kopplung an individuelle Anwendungsstrukturen, Zustände und Managementtechnologien resultiert (vgl. *Software-Alterung* [[Par94](#)]). Neben sich ändernden Managementschnittstellen stellen manuelle Änderungen an laufenden Anwendungen die größte Ursache für fehlerhafte Ausführungen dar, da insbesondere Managementpläne explizite und implizite Annahmen über die Struktur der Anwendung sowie deren gegenwärtigen Zustand und die orchestrierten Managementtechnologien treffen. Die Alterung imperativer Managementpläne kann in zwei Klassen unterteilt werden: *Explizite Alterungseffekte* betreffen syntaktische Änderungen an den Schnittstellen aufzurufender Technologien, wodurch es zu Fehlern bei deren Aufruf und somit zum Scheitern der Prozessausführung kommt. Im Gegensatz dazu belassen *implizite Alterungseffekte* den Prozess als solchen syntaktisch in einer ausführbaren Form, beeinflussen jedoch die Effekte der Ausführung zum Negativen, beispielsweise wenn eine Software auf einer virtuellen Maschine installiert werden soll, die nicht mehr existiert. Da sich Änderungen in vielen Szenarien nicht vermeiden lassen oder unerlaubterweise durchgeführt werden, müssen Managementpläne für den aktuellen Stand einer Anwendung und die involvierten Managementtechnologien generiert werden [[HK10b](#)]: Eine vorab getätigte Modellierung führt zu kontinuierlichen Anpassungen und Wartungsarbeiten an den Prozessmodellen, welche die Gesamtkosten des Anwendungsmanagements deutlich steigern ohne vom Zugewinn neuer Funktionalität zu profitieren.

Die vorliegende Arbeit behandelt diese Problematik durch die automatisierte Generierung von Managementplänen für den gegenwärtigen Zustand einer Anwendung auf Anfrage, d. h. erst wenn die gewünschten Managementaufgaben tatsächlich auszuführen sind. Das Problem der Kopplung und Alterung von Managementplänen ist daher im vorgestellten PALMA-Ansatz nicht existent. Die beschriebenen Probleme bezüglich sich ändernder Schnittstellen führen durch Auslagerung von Aufrufen in Management-Planlets ausschließlich zu lokalen Änderungen, wodurch der Aufwand für nötige Anpassung reduziert wird.

2.3.6 Hybride Managementansätze und Verfeinerungstechniken

In diesem Abschnitt werden Arbeiten vorgestellt, die (i) imperative und deklarative Konzepte integrieren oder (ii) abstrakte Managementanforderungen, die als deklarative Beschreibung der zu erreichenden Ziele verstanden werden können, durch Verfeinerung in imperative Managementanweisungen überführen.

Bellavista et al. [BCFP13] setzen manuell modellierte Workflow-Modelle dazu ein, technologieabstrahierende Webservices zur Provisionierung von SaaS-Anwendungen auf Infrastrukturdiensten (IaaS) zu orchestrieren. In der prototypischen Umsetzung wird BPEL zur Modellierung von Workflows, Juju zur Realisierung der Webservices und OpenStack als IaaS-Schicht verwendet. Durch die vorgestellte Architektur können auch deklarative Managementtechnologien als Webservice gekapselt und in die Provisionierung der Anwendung eingebunden werden. Breitenbücher et al. [BBK+15a] entwickeln dieses Konzept weiter und definieren einen neuen Aktivitätstyp für Workflow-Sprachen, welcher die direkte Einbettung deklarativer Managementbeschreibungen einer bestimmten Technologie, z. B. Puppet, in imperative Workflow-Modelle ermöglicht.

Keller et al. [KHW+04] stellen mit *CHAMPS*¹ ein Managementsystem zur Ausführung von Requests for Change vor. RFCs werden im Change Management eingesetzt, um deklarativ und abstrakt zu beschreiben, welche Änderungen an den Komponenten der Infrastruktur durchgeführt werden sollen. Zum Beispiel, dass eine Geschäftsanwendung auf einem bestimmten Server installiert werden

¹CHAnge Management with Planning and Scheduling

soll (siehe [Abschnitt 2.2.2](#)). Die deklarativ beschriebenen Managementaufgaben werden durch Analyse von Deployment-Deskriptoren und dem Instanzmodell der betroffenen Anwendung zunächst auf die elementaren Schritte heruntergebrochen, partiell geordnet und in ein abstraktes Workflow-Modell (*Task Graph*) übersetzt. Mehrere dieser Task Graphs werden anschließend von einer *Planner and Scheduler*-Komponente konsumiert, welche deren Ausführung optimiert und aufeinander abstimmt. Dazu werden die auszuführenden Aufgaben an logische und physische Ressourcen gebunden und in einen *Change Plan* transformiert, der spezifizierte Policies und Service Level Agreements einhält. Anschließend wird der optimierte Plan von einem *Deployment System* ausgeführt. Da Task Graphs und Change Plans in Form von Workflow-Modellen beschrieben sind, können diese vom zuständigen Administrator adaptiert werden, um die Ausführung zu überwachen und individuelle Anpassungen vorzunehmen. In der vorliegenden Arbeit wird diese manuelle Adaption in gleicher Form ermöglicht, indem deklarative DMMN-Modelle in ausführbare Workflow-Modelle übersetzt werden. In einer weiteren Arbeit [KB04] zeigen die Autoren, wie die Sprache BPEL eingesetzt werden kann, um CHAMPS mit Allzweck-Workflow-Laufzeitumgebungen (engl. „*general-purpose workflow engines*“) und existierenden Bereitstellungssystemen zu integrieren.

Neben dem AI-Planning stellt das *Hierarchical Task Network Planning (HTN-Planning)* [GNT04] den zweiten wissenschaftlich relevanten Planning-Ansatz dar, der auf die Managementdomäne angewendet wurde. Aufgrund dessen imperativer Natur ermöglicht HTN-Planning die Modellierung und Ausführung zustandsmodifizierender sowie zustandserhaltender Managementaufgaben. Trastour et al. [TFL09] stellen dazu mit *ChangeRefinery* ein Assistenzsystem zur Beschreibung von Requests for Change (siehe [Abschnitt 2.2.2](#)) vor, das die Automatisierung und strukturierte Erfassung existierenden Managementwissens und „Best Practices“ ermöglicht. Dazu werden häufig wiederkehrende, technologieunabhängig formulierte High-level Managementaufgaben mit vorsezifzierten *Hierarchical Task Networks* verknüpft, die eine hierarchische, temporal geordnete Strukturierung auszuführender Managementaktivitäten auf unterschiedlichen Abstraktionsebenen beschreiben. Diese können vom Be-

nutzer, in diesem Fall dem *Change Requester*, aus einem Katalog ausgewählt, parametrisiert und automatisiert mittels HTN-Planning für eine bestimmte Anwendung in auszuführende atomare Operationen verfeinert werden.

HTN-Planning wurde von Hagen et al. [HEW+09] mit AI-Planning zu einem hybriden Ansatz integriert. Managementaufgaben werden in Form von *Change Requests (CR)* spezifiziert, die entweder (i) atomare Operationen darstellen, (ii) noch mittels HTN-Planning zu feingranulareren CRs verfeinert werden müssen oder (iii) Zielzustände spezifizieren, die durch AI-Planning wiederum in CRs übersetzt werden, deren Ausführungen diese Zustände erreichen. Durch dieses Metamodell und die zugehörigen Algorithmen können Change Requests deklarativ sowie imperativ spezifiziert und vollautomatisiert solange rekursiv verfeinert werden, bis ausschließlich atomare, ausführbare Operationen resultieren. Die Autoren stellen in einer späteren Arbeit zudem Optimierungstechniken für das *Simple Task Network Planning*¹ vor, um die automatisierte Verfeinerung von Change Plans in ausführbare Workflow-Modelle für große, komplexe Anwendungsstrukturen zu ermöglichen [HdCG+12].

Speziell die Arbeiten von Hagen et al. [HEW+09; HdCG+12] zeigen auf, wie das deklarative und das imperative Managementparadigma miteinander integriert werden können. Dadurch können zustandsmodifizierende Managementaufgaben gemäß des etablierten Modelltyps der Zielzustandsmodelle umgesetzt werden, während zustandserhaltende Managementaufgaben durch eine imperative Modellierungsform realisierbar sind. Im Gegensatz zum rekursiven Ansatz von Hagen et al. wird in der vorliegenden Arbeit mit DMMN eine Sprache vorgestellt, in der beide Typen von Managementaufgaben direkt deklarativ modellierbar sind. Dadurch kann der Administrator explizit vorgeben, welche Aufgaben ausgeführt werden sollen. Durch die Transformation in imperative Managementpläne werden die beiden Managementparadigmen in der vorliegenden Arbeit ineinander überführt, jedoch nicht miteinander integriert. Die Modellierung deklarativer Ziele in imperativen Prozessmodellen, d. h. die Integration beider Paradigmen, ist Gegenstand zukünftiger Forschungsarbeiten.

¹Eine vereinfachte Form des HTN-Plannings, bei der durch Vorwärtsverkettung (engl. „forward-chaining“) und Tiefensuche eine Verfeinerung des Plans berechnet wird [HdCG+12]

2.3.7 Risikomanagement, Fehlerprävention und Fehlerbehandlung

Informationstechnologie ist in vielen Organisationen ein fundamentaler Bestandteil für die Erbringung der Geschäftsergebnisse. Der Ausfall von Geschäftsanwendungen ist meist direkt mit finanziellen oder strategischen Einbußen verbunden, weswegen Managementfehler bezüglich dieser Anwendungen nicht tolerabel sind. Um ungewollte Effekte und Fehler präventiv zu vermeiden, definiert beispielsweise ITIL einen expliziten Schritt zur Analyse möglicher Risiken hinsichtlich der durchzuführenden Managementaufgaben, bevor die Änderungen tatsächlich an der IT vorgenommen werden [WBL+09].

Komplexe Anwendungsstrukturen erschweren dieses *Risikomanagement* jedoch signifikant: Soll eine Änderung an einer Komponente durchgeführt werden, können dem Administrator unbekannte Abhängigkeiten zu anderen Komponenten deren korrekte Funktionsweise gefährden [KBK00]. Wird beispielsweise eine Datenbank deinstalliert, welche fälschlicherweise als ungenutzt erachtet wurde, können darauf zugreifende Komponenten ihre Funktion nicht mehr erbringen. Häufig können diese Art der Managementfehler nur unter großem Zeitaufwand rückgängig gemacht werden [XZL+13]. Beim Anwendungsmanagement spielt somit (i) das Verständnis der aktuellen Anwendungsstruktur und deren Abhängigkeiten sowie (ii) die Konfiguration der involvierten Komponenten eine fundamentale Rolle [AEK+07]. Ist dieses Wissen nicht vorhanden oder wird ignoriert, kann ein fehlerfreies Anwendungsmanagement nicht gewährleistet werden [Opp03]. Insbesondere sind manuelle Fehler ein negativer Einflussfaktor, den es zu eliminieren gilt [OGP03; Opp03]. Die automatisierte Ausführung von Managementaufgaben kann jedoch aufgrund von unbekannten Abhängigkeiten zwischen Komponenten und deren Konfigurationen ebenfalls zu Problemen führen, etwa wenn mehrere Managementtechnologien kombiniert werden [BBK+14b]. Während Ansätze zur Echtzeiterkennung aufgetretener Fehler existieren, wie beispielsweise von Xu et al. [XWB+13] vorgestellt, stellt die Fehlervermeidung aufgrund gewachsener Prozesse eine signifikante Herausforderung dar. Unabhängig davon, ob Management manuell oder automatisiert durchgeführt wird, werden Risiken in den meisten Fällen durch menschliche Administratoren abgeschätzt, was aufgrund der enormen Komplexität involvierter

Komponententypen und Managementtechnologien eine unzuverlässige Ausgangslage darstellt [WBL+09]. Daher ist eine Kernanforderung an die Konzepte der vorliegenden Arbeit, dass der *Kontext*, in dem Managementaufgaben ausgeführt werden, (i) explizit erfasst und (ii) vor der Durchführung automatisiert auf mögliche Fehler und Risiken analysiert werden muss.

Sowohl die existierenden imperativen als auch teilweise die deklarativen Ansätze können jedoch in vielen Fällen nicht den gesamten Kontext der Managementdurchführung darstellen, wodurch eine automatisierte Risikoanalyse hinsichtlich möglicher Fehler weder konzeptionell noch technisch möglich ist: Imperative Ansätze beschreiben typischerweise ausschließlich auszuführende Anweisungen, jedoch nicht den Gesamtkontext, in dem diese ausgeführt werden [BBK+14b]. So kennt ein imperatives Skript beispielsweise nur in den seltensten Fällen die virtuelle Maschine, auf der es ausgeführt wird, oder andere Programme, die auf diese virtuelle Maschine zugreifen und möglicherweise von Änderungen an der Konfiguration der Maschine beeinflusst werden. Ob die Ausführung eines Skripts andere Anwendungskomponenten negativ beeinflusst, kann somit generell nicht durch Analyse des Skriptes allein verifiziert werden, sondern erfordert zusätzliche Informationen, die im Skript selbst typischerweise nicht enthalten sind. Gleiches gilt ebenso für andere imperative Ansätze, wie beispielsweise Workflows. Deklarative Ansätze leiden zum Teil unter demselben Problem, etwa wenn Zielzustandsmodelle nur die zu modifizierenden Komponenten beschreiben, jedoch nicht die gesamte Anwendung inklusive aller Relationen zwischen den potenziell betroffenen Anwendungskomponenten. Um Managementfehler zu vermeiden bzw. zu behandeln, werden in den folgenden beiden Abschnitten präventive und reaktive Ansätze vorgestellt.

2.3.7.1 Fehlerprävention

Zur Reduktion bzw. ganzheitlichen Vermeidung von Managementfehlern wurden *präventive* Ansätze zur Risiko- und Fehleranalyse entwickelt. Wickboldt et al. [WBL+09; WMdC+09; BWG+10] stellen ein Verfahren vor, um das Risiko möglicherweise auftretender Fehler bezüglich einer Managementaktivität innerhalb eines Managementplans durch Untersuchung früherer Ausführungen

ähnlicher Managementpläne abzuschätzen. Dazu wird für jede Aktivität analysiert, wie oft in der Vergangenheit ein bestimmter Fehler für die spezifizierte Managementaufgabe bezüglich der betroffenen Komponente aufgetreten ist. Daraus wird die Auftrittswahrscheinlichkeit für den zu untersuchenden Plan berechnet. Im Gegensatz dazu definieren Fu et al. [FZBX14] generelle Kriterien, um Operationen in einem Managementprozess auf Wiederherstellungsmöglichkeiten im Falle auftretender Fehler zu analysieren und diesbezüglich zu überarbeiten (engl. „*Recoverability-Oriented Analysis*“).

Hagen et al. [HSK12; HdCG+14] stellen eine objektorientierte Verifikationstechnik vor, um präventiv Konflikte zwischen auszuführenden IT-Änderungsaufgaben (engl. „*IT Change Operations*“) und spezifizierten Sicherheitsbeschränkungen (engl. „*Safety Constraints*“) in linearer Zeit zu erkennen. Dadurch kann sichergestellt werden, dass eine Sicherheitsbeschränkung, z. B. eine maximale Kapazitätsbeschränkung eines Netzwerk-Routers, nicht durch eine geplante Änderung an der IT-Konfiguration verletzt wird. Der Ansatz basiert auf der formalen Spezifikation der Vorbedingungen und Effekte von Change Operations bezüglich der betroffenen Komponenten, auf Basis derer Verletzungen der formal spezifizierten Safety Constraints berechnet werden können. Diese Form der Formalisierung nutzen Hagen und Kemper in einer anderen Arbeit [HK11], um unabhängig voneinander erstellte Change Plans auf Konflikte zu untersuchen: Bevor ein Plan ausgeführt wird, werden die Vorbedingungen und Effekte aller darin spezifizierten Aufgaben präventiv mit aktuell ausgeführten Plänen verglichen und analysiert, ob diese Konflikte zueinander aufweisen.

Einige Ansätze setzen bewährte formale Konzepte und Algorithmen zur präventiven Vermeidung von Fehlern ein, indem auszuführende Managementaufgaben bzw. Zielzustandsmodelle als Probleme formuliert werden. Hewson et al. [HAG12] ermöglichen beispielsweise die Erkennung von Konfigurationsproblemen durch Formulierung der gewünschten Systemkonfiguration als Constraint Satisfaction Problem (CSP) und dessen Lösung (siehe [Abschnitt 2.3.3](#)).

In der vorliegenden Arbeit wird die präventive Erkennung von Managementfehlern durch ein musterbasiertes Konzept automatisiert, welches die kontext-sensitive Analyse der in DMMN-Modellen deklarieren Managementaufgaben

hinsichtlich Fehlern ermöglicht und diese automatisiert auf Basis bewährten Managementwissens korrigiert ([Forschungsbeitrag 5](#)). Dadurch wird der Administrator bei der korrekten Spezifikation bzw. Durchführung von Managementaufgaben unterstützt und die dazu benötigte Expertise reduziert. Die gleichzeitige Ausführung mehrerer Managementprozesse und die dazu nötige Konfliktvermeidung sind nicht Forschungsgegenstand der vorliegenden Arbeit, können jedoch durch die Ansätze von Hagen und Kemper [[HK11](#)] auf die vorgestellten Forschungsbeiträge übertragen werden. Ebenso wird die Optimierung der Ablaufplanung von Managementaufgaben (engl. „*scheduling*“) nicht gesondert betrachtet. Dazu existierende Ansätze, wie beispielsweise von Rebouças et al. [[RSM+07](#)], Trastour et al. [[TRB07](#)] und Dumitraş et al. [[DRDN07](#)] vorgestellt, werden in zukünftigen Arbeiten in die PALMA-Methode integriert.

2.3.7.2 Fehlerbehandlung

Dem gegenüber stehen *reaktive* Ansätze, welche die Behandlung aufgetretener Fehler fokussieren. Machado et al. [[MDdC+08](#)] stellen einen Ansatz vor, um Requests for Change (RFCs, siehe [Abschnitt 2.2.2](#)), deren Operationen oder die Aktivitäten der jeweils zugehörigen Change Plans als atomare Transaktionen zu annotieren. Scheitert die Ausführung eines dieser annotierten Elemente, werden automatisch generierte *Rollback Plans* ausgeführt, welche reaktiv die getätigten Änderungen des partiell ausgeführten Change Plans rückgängig machen und den vorherigen, konsistenten Anwendungszustand wiederherstellen (engl. „*recovery*“). Die Generierung von Rollback Plans basiert auf der Umkehrung der Ausführungsreihenfolge und dem Aufruf der jeweils kompensierenden Operationen. In einer späteren Arbeit wurde dieses Konzept verfeinert, um die Spezifikation individueller Fehlerbehandlungs- und Kompensationslogik für die annotierten Elemente in Form sogenannter *Remediation Plans* zu ermöglichen, wodurch Fehler feingranularer, differenzierter und flexibler gehandhabt werden können [[MdCdS+09](#)]. In der vorliegenden Arbeit werden auftretende Fehler, die nicht präventiv erkannt bzw. vermieden werden können, ähnlich der letztgenannten Arbeit durch explizite Fehlerbehandlungs- und Kompensationslogik gelöst, welche direkt im generierten Managementplan enthalten ist.

Ein Managementplan stellt somit eine atomare Transaktion dar, die entweder vollständig ausgeführt oder im Falle eines Fehlers in ihrer Gesamtheit zurückgesetzt wird. Da Managementpläne in dieser Arbeit aus Management-Planlets generiert werden, stellen diese individuelle Fehlerbehandlungs- und Kompensationslogik bereit, welche vom generierten Managementplan zur Rücksetzung aller Änderungen aufgerufen werden kann (siehe [Abschnitt 5.2.4](#)).

Ähnlich dazu führen Cuadrado et al. [[CGND10](#)] ein Metamodell zur Beschreibung imperativer Managementpläne ein, das generische, atomare Aktivitätstypen für Managementaufgaben definiert. Tritt während deren Ausführung ein Fehler auf, wird die Ausführung des Plans abgebrochen, noch nicht ausgeführte Aktivitäten entfernt, Kompensationsaktivitäten in den Plan eingefügt und die Ausführung des adaptierten Plans fortgesetzt, um alle Änderungen rückgängig zu machen. Kompensationsaktivitäten werden durch Umkehrung der jeweiligen Operationsaufrufe der bereits erfolgreich ausgeführten Aktivitäten berechnet.

Viele Managementtechnologien bieten jedoch nicht für jede Operation eine dedizierte Umkehroperation an, welche zur Rücksetzung der durch die Operation durchgeführten Änderung bzw. den durch diese ausgelösten Effekt aufgerufen werden kann. Häufig erfordert eine solche Rücksetzung die Ausführung mehrerer generischer Operationen. Muss beispielsweise eine gelöschte virtuelle Maschine wiederhergestellt werden, erfordert dies eine Vielzahl unterschiedlicher Operationsaufrufe inklusive der Installation aller benötigten Softwarepakete etc. Weber et al. [[WWF+12](#)] stellen dazu ein Konzept vor, um Aufrufe von Managementoperationen an Cloud-APIs rückgängig zu machen. Die Rücksetzung der getätigten Änderungen, d. h. das *Rollback* der aufgerufenen Managementoperationen, basiert auf der Erfassung des Anwendungszustandes vor Ausführung der Operationen, welcher im Falle eines Rollbacks als Zielzustandsmodell für einen AI-Planner verwendet wird, um einen Kompensationsplan zu generieren. Es wird zudem eine Systemarchitektur eingeführt, die APIs kapselt (engl. „*wrapping*“), um *nicht-reversible* Operationen, d. h. Operationen, für welche das API keine Umkehroperation anbietet, als *reversible* Operationen zu exponieren. Um die Löschung einer virtuellen Maschine innerhalb eines umfangreicheren Managementprozesses effizient rückgängig zu machen, wird diese Operation

beispielsweise erst dann real durchgeführt, wenn alle anderen Operationsaufrufe des Plans erfolgreich abgearbeitet wurden (*pseudo-delete* [Gra10]). Im Falle eines Prozessabbruchs wird der durch das Managementsystem simulierte Zustand der gelöschten Maschine verworfen, wodurch die Rücksetzung keine erneute Provisionierung und Konfiguration erfordert. In einer späteren Arbeit [WWF+13] wird dieses Konzept von denselben Autoren erweitert, um Operationen präventiv daraufhin zu analysieren, zu welchem Grad deren Ausführung anschließend wieder rückgängig gemacht werden kann.

Die Ansätze von Cuadrado und Weber unterscheiden sich zum Konzept der vorliegenden Arbeit dahingehend, dass Management-Planlets eigene, vordefinierte Kompensationslogik spezifizieren und native Mechanismen der eingesetzten Workflow-Sprache zur Fehlerbehandlung bzw. Kompensation verwendet werden (siehe Abschnitt 5.2.4 und Abschnitt 5.5.1). Eine Berechnung von Kompensationsprozessen zur Laufzeit ist daher nicht erforderlich. Vor- und Nachteile dieses Ansatzes werden in Abschnitt 5.6 diskutiert.

Arshad et al. [AHW03; AHW07] stellen einen Replanning-Ansatz vor, um aufgetretene Fehler bei der Konfiguration von Anwendungen durch eine automatische Generierung eines neuen Zielzustandsmodells zu behandeln. Das generierte deklarative Modell wird anschließend automatisiert in einen ausführbaren, imperativen Managementplan übersetzt, der die Anwendung in einen konsistenten Anwendungszustand zurückführt. Im Gegensatz dazu werden Fehler im Rahmen der PALMA-Methode sofort durch Rücksetzung aller Änderungen durch den ausgeführten Managementplan selbst behandelt.

2.3.8 Zusammenfassung und Diskussion

Dieser Abschnitt fasst die Erkenntnisse der in diesem Kapitel vorgestellten imperativen und deklarativen Managementkonzepte zusammen und bietet somit die Grundlage einer umfassenden Anforderungsanalyse für die PALMA-Methode, die in [Abschnitt 3.2](#) vorgestellt wird.

Ein wesentliches Unterscheidungsmerkmal zwischen den vorgestellten Ansätzen ist deren Unterstützung zustandserhaltender und zustandsmodifizierender Managementaufgaben. Während deklarative Zielzustandsmodelle intuitiv dazu geeignet sind die Struktur und Konfiguration einer Anwendung zu verändern, wird die deklarative Modellierung zustandserhaltender Aufgaben von den existierenden Ansätzen nicht unterstützt. Einige Ansätze ermöglichen die manuelle Anpassungen generierter Workflow-Modelle, um diesen Aufgabentyp zu automatisieren. Dies ist jedoch für umfangreiche Managementaufgaben und komplexe Anwendungen nicht effizient umsetzbar, da imperative Ansätze generell enormen Aufwand erfordern, um die nötigen Prozessmodelle zu erstellen. Im Vergleich dazu ist die deklarative Modellierung aufgrund der höheren Abstraktion deutlich effizienter und weniger fehleranfällig durchführbar. Die deklarative Modellierung ist daher diesbezüglich einfacher und erfordert weniger Expertise, weswegen sich dieses Paradigma deutlich besser zur Interaktion mit dem Administrator eignet. Zudem ist das Resultat der Ausführung direkt in deklarativen Modellen ersichtlich, während imperative Modelle typischerweise Anweisung für Anweisung nachvollzogen werden müssen, um das finale Ergebnis bzw. den finalen Zustand der Anwendung zu berechnen [[HAW11](#)]. Dadurch wird die Überprüfung der Modelle auf Fehler signifikant erschwert.

Die Beeinflussbarkeit der Managementdurchführung ist ein grundlegendes Kriterium, da komplexe Szenarien häufig eine individuelle, anwendungsspezifische Managementlogik erfordern. Deklarative Ansätze sind im Vergleich zu imperativen Ansätzen weniger beeinflussbar, da sie nur in vordefinierten Standardphasen eine Individualisierung der Managementdurchführung ermöglichen. Zudem müssen die tatsächlich auszuführenden Anweisungen in deklarativen Ansätzen aus dem Modell abgeleitet bzw. dieses interpretiert werden, wodurch

Ausführungsumgebungen meist an vordefinierte Typen, Schnittstellen und Eigenschaften von Komponenten gebunden sind. Des Weiteren ist die Ausführung deklarativer Modelle vom jeweils eingesetzten Interpreter bzw. Transformationsverfahren abhängig, wodurch die Portabilität der Modelle nicht ohne die explizite Spezifikation der Semantik aller Modellbestandteile, z. B. Komponententypen, gewährleistet werden kann. Da Managementlogik nicht beliebig spezifiziert werden kann, eignen sich deklarative Technologien vorwiegend zur Bereitstellung von Anwendungen, die eine geringe Komplexität aufweisen und verbreitete Standardkomponenten (engl. „*general-purpose components*“) einsetzen, sowie zur Modellierung zustandsmodifizierender Konfigurationsaufgaben. Da imperative Ansätze beliebige Managementlogik abbilden können, eignen sich diese Technologien zur Bereitstellung und dem Management komplexer Anwendungen, welche die Orchestrierung verschiedenster Managementtechnologien erfordern. Aufgrund der Tatsachen, dass die manuelle Bearbeitung imperativer Prozessmodelle nicht effizient umsetzbar ist und deklarative Technologien die Beeinflussung nur limitiert erlauben, existiert gegenwärtig keine Technologie, die dem Administrator eine intuitive und effiziente Beeinflussung der Managementdurchführung erlaubt.

Die imperative Workflow-Technologie eignet sich aufgrund ihrer Robustheit zur verlässlichen Ausführung komplexer Managementprozesse. Müssen kritische Managementaufgaben ausgeführt werden, welche die Anwendung bei fehlerhafter Durchführung oder einem Abbruch in einen inkonsistenten Zustand führen, sind weder die manuelle Managementdurchführung noch andere Technologien, die keine expliziten Mechanismen zur Absicherung der Durchführung implementieren, dazu geeignet. Zudem können automatisiert ausgeführte Workflows überwacht werden, wodurch die Nachvollziehbarkeit der tatsächlich ausgeführten Änderungen gewährleistet wird. Dies ist vor allem in rechtlichen Fragestellungen häufig von grundlegender Bedeutung. Die Workflow-Technologie ist daher als Grundlage zur Orchestrierung verschiedener Managementtechnologien geeignet, weswegen diese bei der Umsetzung der PALMA-Methode die Basis zur Ausführung von Managementprozessen darstellt.

2.4 Muster in der Informationstechnologie

Der Begriff „Muster“ (engl. „*pattern*“), wie er heute in verschiedenen Domänen der Informationstechnologie verwendet wird, wurde in den siebziger Jahren durch den Architekten Christopher Alexander im Bereich des Hausbaus und der Stadtplanung geprägt [AIS77; Ale79]. Ein Muster beschreibt demnach eine bewährte Lösung eines häufig in einem bestimmten Kontext auftretenden Problems in abstrakter Form, wodurch konzeptionelles Wissen, „Know-How“ und „Best Practices“ generalisiert werden können. Dadurch kann bewährtes Wissen unabhängig von konkreten Technologien und spezifischen Anwendungsfällen dokumentiert, wiederverwertet und angewendet werden [Fow02]. Der Abstraktionsgrad verschiedener Muster variiert dabei in der Praxis zum Teil stark, wodurch einige Muster durch Abstraktion und Generalisierung ausschließlich die konzeptionelle Essenz eines Problems und der zugehörigen Lösung erfassen, andere Muster hingegen detaillierte Problem- und Lösungsbeschreibungen für spezifische Anwendungsfälle dokumentieren [FBB+15]. Der Begriff „Muster“ wird daher für die vorliegende Arbeit folgendermaßen informell definiert:

Definition 9 (Muster - informell). Ein Muster dokumentiert eine generische, wiederverwendbare und bewährte Lösung eines häufig auftretenden Problems in einem bestimmten Kontext. ■

Muster bestehen typischerweise aus verschiedenen textuellen Bestandteilen, welche unter anderem das Problem, die Lösung sowie die Herausforderungen in natürlichsprachlichem Text beschreiben. Einige Muster dokumentieren zudem Beispiele, in denen das Muster erfolgreich angewendet wurde, und verweisen auf andere Muster, mit denen es kombiniert werden kann. Eine Übersicht unterschiedlicher Bestandteile von Mustern wird von Ernst [Ern10] gegeben.

Um zusammenhängende Muster verschiedener Problemstellungen einer bestimmten Domäne in einer einheitlichen Beschreibungsstruktur zu erfassen, werden verwandte Muster in Form einer *Mustersprache* (engl. „*pattern language*“) dokumentiert. Mustersprachen beschreiben somit den semantischen Zusammenhang der einzelnen Muster [AIS77; FLR+14]. Buschmann [BMR+96]

nennt diese Mustersprachen auch *Mustersysteme* (engl. „*pattern systems*“), da der Begriff „Sprache“ suggeriert, dass die enthaltenen Elemente eine bestimmte Domäne vollständig beschreiben, was dem möglichen Erkenntniszugewinn in Form neu identifizierter Muster innerhalb einer bereits dokumentierten Domäne widerspricht. Mustersprachen können auch formal durch Grammatiken beschrieben werden, um die Abhängigkeiten zwischen den Mustern innerhalb der Sprache zu erfassen [Zdu07].

Mustersprachen werden bei der Entwicklung sowie dem Management von IT-Systemen in verschiedenen Bereichen eingesetzt: Architekturmuster (engl. „*Architecture Patterns*“) [BMR+96; Fow02; AZ05] unterstützen die fundamentale Organisation komplexer Systeme, welche durch Entwurfsmuster (engl. „*Design Patterns*“) [GHJV94; BMR+96] technisch realisiert werden können. Zudem können Architekturmuster auch die Dokumentation von Architekturentscheidungen unterstützen [HAZ07]. Zur Entwicklung von Cloud Anwendungen wurden *Cloud Computing Patterns* [FLR+11; Wil12; FEL+12; FLR+14] und *Cloud Data Patterns* [SBK+12; SAB+12; SAB+13; SABL13] identifiziert. Integrationsmuster lösen Probleme, die bei der Integration verschiedener Anwendungssysteme auftreten [TRH+04], beispielsweise durch die *Enterprise Integration Patterns (EIP)* [HW04]. Weitere verbreitete Mustersprachen stellen *Implementation Patterns* [Bec96; Bec07], *Workflow Patterns* [vABtHK00; vAtHKB03], *Service Interaction Patterns* [BDtH05], *Green Business Process Patterns* [NLS+11; NL13; Now14], sowie die *Process Viewing Patterns* [SLS10; Sch15] dar.

Zur Dokumentation und Erfassung von Mustern wurden zahlreiche Techniken und Methoden entwickelt [MD97; Har99; LdAA+04; FLR+11; WF12; FEL+12; Rei12; RHEP12]. Fehling et al. [FBBL14] stellen beispielsweise ein detailliertes Vorgehensmodell vor, um die (i) Identifikation, (ii) Extraktion und (iii) Anwendung beliebiger Mustersprachen zu unterstützen und die Aktivitäten beteiligter Personen zu koordinieren. Um die kollaborative Dokumentation, Verbindung und Suche nach Mustern durch Werkzeuge zu unterstützen, werden häufig Wikis [LC01; Sal05] bzw. Wiki-basierte Werkzeuge, wie *PatternPedia* [FBFL15; FL14] oder das *Open Pattern Repository* [vHee14], eingesetzt.

2.4.1 Managementmuster

Die vorliegende Arbeit adressiert die Automatisierung von Managementmustern, die Managementprobleme durch Beschreibung bewährter Managementprozesse lösen, z. B. die Managementprozessmuster für Cloud-Anwendungen [FLRS12; FLR+14], die Dienstmigrationsmuster [FLR+13] sowie die Muster zur Migration von Komponenten in Cloud-Umgebungen von Jamshidi et al. [JPCL14].

Ein Beispiel ist das *Stateless Component Swapping-Muster* [FLR+13], das folgendes Problem löst: „Wie können zustandslose Anwendungskomponenten ohne Ausfallzeit migriert werden?“ Der Kontext dieses Musters ist dadurch gegeben, dass der Ausfall einer Anwendung in vielen Geschäftsszenarien nicht akzeptabel ist. Eine zustandslose Anwendungskomponente soll daher *transparent* für zugreifende Entitäten, d. h. Menschen oder andere Anwendungen, migriert werden, sodass diese von der Migration nicht negativ beeinflusst werden. Zur Lösung dieses Problems beschreibt das Muster folgenden Prozess: (i) Zuerst muss die Implementierung der zu migrierenden Komponente aus der Ursprungsumgebung extrahiert werden, während gleichzeitig die Zielumgebung konfiguriert wird. (ii) Anschließend wird diese Implementierung in der Zielumgebung bereitgestellt, während die ursprüngliche Komponente ungeachtet dessen weiter betrieben wird. (iii) Im dritten Schritt werden alle Zugriffe auf die ursprüngliche Komponente dauerhaft auf in die neue Zielumgebung, d. h. auf die neu bereitgestellte Komponente, umgeleitet. (iv) Abschließend wird die ursprüngliche Komponente außer Betrieb genommen [FLR+13]. Derart Managementmuster unterstützen den Administrator somit bei der Managementdurchführung, indem sie die auszuführenden Schritte konzeptionell vorgeben.

Definition 10 (Managementmuster- informell). Ein Managementmuster dokumentiert (i) eine generische, wiederverwendbare und bewährte Lösung eines (ii) häufig auftretenden Managementproblems in (iii) einem bestimmten Kontext in Form eines auszuführenden Managementprozesses. ■

Der Abstraktionsgrad unterschiedlicher Managementmuster kann dabei deutlich variieren, sodass neben abstrahiertem Wissen auch technologiespezifische Expertise durch das Konzept dokumentiert werden kann.

2.4.2 Automatisierung von Managementmustern

In diesem Abschnitt wird die Anwendung von Managementmustern zum Management laufender Anwendungen diskutiert und dabei unterschiedliche Möglichkeiten zu deren Automatisierung vorgestellt und evaluiert.

2.4.2.1 Anwendung und Automatisierung von Musterlösungen

Managementmuster dokumentieren Lösungen häufig wiederkehrender Managementprobleme in Form generisch beschriebener Managementprozesse, wie insbesondere von Fehling et al. [FLRS12; FLR+13; FLR+14] mittels BPMN-Modellen zur Lösung von Managementproblemen für Cloud-Anwendungen gezeigt. Beispielsweise, um durch Anwendung des *Update Transition Process-Musters* [FLR+14] eine Anwendungskomponente ohne Ausfallzeit auf eine neue Version zu aktualisieren oder das *Stateless Component Swapping-Managementmuster* [FLR+13] zur Migration einer Anwendungskomponente. Durch eine generalisierte, abstrakte Beschreibung der Managementprozesse können diese Managementmuster das Lösen von konkreten Problemen durch Dokumentation bewährter „Best Practices“ unterstützen, welche anschließend als Vorlage zur manuellen Anwendung und Verfeinerung hinsichtlich bestimmter Laufzeitumgebungen bzw. bestimmter Anwendungsfälle dienen. Da eine manuelle Managementdurchführung aufgrund der Diskussion in Abschnitt 2.2.3.1 nicht effizient umgesetzt werden kann, überträgt sich dieses Problem inhärent auf die manuelle Ausführung der beschriebenen Prozesse dieser Managementmuster.

Einen möglichen Lösungsansatz stellt die Implementierung der Musterlösung in Form eines ausführbaren Managementplans dar, um die Ausführung der Lösung für einen konkreten Anwendungsfall zu automatisieren. Die manuelle Implementierung resultiert jedoch in einigen Herausforderungen:

(i) Zum einen müssen abstrakt beschriebene Musterlösungen für konkrete Anwendungsfälle verfeinert werden [FBB+14b]. In direkter Abhängigkeit zur Komplexität der Anwendung, für die das Muster angewendet werden soll, müssen die abstrakt beschriebenen Managementaufgaben auf die jeweiligen Komponenten der Anwendung und die einzusetzenden Managementtechnologien

en abgebildet werden [BBKL14c]. Dieser Schritt erfährt zusätzliche Komplexität, wenn eine imperativ beschriebene, abstrahierte Lösung eines Managementmusters in technische Aufrufe deklarativer Werkzeuge übersetzt werden muss.

(ii) Zum anderen erfordert die manuelle Implementierung von Managementplänen, wie in [Abschnitt 2.3.5](#) diskutiert, enorme technische Expertise bezüglich der involvierten Managementtechnologien und ist aufgrund der hohen Komplexität hinsichtlich deren technischer Orchestrierung zu einem Prozessmodell eine fehleranfällige, ineffiziente und kostenintensive Vorgehensweise [BBKL14c]. Da alle Eventualitäten der betroffenen Anwendung, auf welche das Muster angewendet wird, im Prozessmodell beachtet und modelliert werden müssen, führt diese Vorgehensweise zu sehr komplexen Prozessmodellen, welche die Wiederverwendung der implementierten Managementlogik erschwert.

(iii) Zudem müssen diese Lösungsimplementierungen individuell jeweils für genau eine konkrete Anwendung vorab erstellt werden, da Managementpläne typischerweise eng an eine bestimmte Anwendungsstruktur und deren Zustand gekoppelt sind (siehe [Abschnitt 2.3.5.2](#)). Dies führt zu einer Vielzahl an zu implementierenden Managementplänen, die zur Automatisierung eines einzelnen Musters erforderlich sind, da unterschiedliche Anwendungsstrukturen, Komponententypen, Laufzeitumgebungen und Anwendungszustände beachtet werden müssen. Daraus resultiert ein enormer Implementierungsaufwand, der vorab investiert werden muss [HAW11]. Werden die automatisierten Musterlösungen während der jeweiligen Anwendungslebenszeit nicht benötigt, wurde dieser Aufwand jedoch unnötigerweise investiert. Zudem unterliegen Managementpläne typischerweise einem Alterungseffekt, wodurch die erstellten Prozessmodelle gewartet und gegebenenfalls aufgrund geänderter Infrastrukturen, APIs der Managementtechnologien oder manuellen Änderungen an der Anwendung angepasst werden müssen (vgl. [Abschnitt 2.3.5.3](#)).

Falkenthal et al. [FBB+14b; FBB+14a] stellen mit den *Solution Implementations* einen Ansatz vor, um wiederverwendbare, direkt ausführbare Lösungsimplementierungen für einen spezifischen Anwendungsfall eines bestimmten Musters mit diesem zu verbinden. *Solution Implementations* sind aggregierbare Bausteine, welche in der Informationstechnologie beispielsweise als ausführbarer

Programmcode oder Workflow-Modelle umgesetzt werden können. Der Ansatz erlaubt es, Managementmuster durch ausführbare Workflow-Modelle für spezifische Anwendungsstrukturen zu implementieren, um die Musteranwendung dadurch für Anwendungen, welche dieser Struktur entsprechen, zu automatisieren. Fehling et al. [FLRS12] schlagen vor, Cloud-Architekturmuster mit konkreten Anwendungsmodellen und ausführbaren Implementierungen, z. B. Prozessmodellen, zugehöriger Managementoperationen zu annotieren. Die Verwendung dieser Implementierungen unterstützt die praktische Umsetzung abstrakter Managementmuster, welche die jeweiligen Architekturmuster als Vorbedingung definieren. Während die manuelle Kombination dieser Implementierungen zur Umsetzung der Musterlösung gemäß [Abschnitt 2.2.3.1](#) nicht effizient ist, können diese auch durch automatisiert ausführbare Prozessmodelle orchestriert werden. Die Nachteile der Ansätze von Fehling und Falkenthal entsprechen jedoch den in diesem Abschnitt und in [Abschnitt 2.3.5](#) diskutierten Problemen der manuellen Prozessmodellierung, da die Prozessmodelle vorab implementiert werden müssen, um bei Bedarf ohne signifikante Verzögerung ausgeführt werden zu können. Aufgrund der starken Kopplung von Managementplänen an Anwendungsstrukturen und Managementtechnologien eignen sich die beiden Ansätze daher lediglich für einfache Managementmuster und verbreitete Anwendungsfälle, die häufig in sehr ähnlicher Form auftreten und nur eine überschaubare Menge an verschiedenen Technologien involvieren, nicht zum Management komplexer Geschäftsanwendungen.

Zusammenfassend kann somit der Schluss gezogen werden, dass sowohl (i) die manuelle Ausführung einer Musterlösung sowie die (ii) manuelle Implementierung der Musterlösung in Form eines ausführbaren Prozessmodells nicht effizient umsetzbar sind. Speziell in Bereichen, die eine unmittelbare Ausführung der Managementprozesse und somit eine Vorabimplementierung der zugehörigen Prozessmodelle erfordern, sind der erforderliche Zeitaufwand und die damit verbundenen Kosten nicht akzeptabel (vgl. [Abschnitt 2.2.3.2](#)). In der vorliegenden Arbeit wird daher eine Methode vorgestellt, welche die automatische Generierung von Managementplänen ermöglicht, um die Anwendung von Managementmustern zu automatisieren.

2.4.2.2 Automatisierung von Mustern in anderen IT-Domänen

Die Automatisierung von Mustern erfährt in verschiedenen IT-Domänen großes Interesse. Dieser Abschnitt stellt verschiedene Ansätze vor und diskutiert deren Eignung zur Automatisierung von Managementmustern.

Chee et al. [CZM+11] stellen ein Verfahren vor, das Architekten und Entwickler dabei unterstützt, eine existierende Anwendung in eine Cloud-Umgebung zu migrieren und diese dafür zu optimieren. Dazu wird das Konzept der sogenannten *Enablement Patterns* vorgestellt, das ähnlich zu Entwurfsmustern (engl. „*Design Patterns*“ [GHJV94]) dabei hilft, Erfahrungen und „Best Practices“ bei der Entwicklung von Anwendungen zu dokumentieren. Enablement Patterns fokussieren dabei auf Cloud-Eigenschaften, wie beispielsweise Security-Anforderungen, welche durch die Muster erzielt werden können und definieren dazu grobgranulare Vorbedingungen, die von der zu migrierenden Anwendung erfüllt sein müssen, um das Muster anzuwenden. Zur Auswahl geeigneter Enablement Patterns werden deren Vorbedingungen und Eigenschaften mit der Anwendung und den gestellten Anforderungen verglichen. In der vorliegenden Arbeit wird zur Auswahl geeigneter Managementmuster ein ähnlicher Ansatz gewählt, jedoch nicht auf Basis des grobgranularen Vergleichs von Anforderungen, sondern durch explizite Beachtung der Anwendungsstruktur, die von einem Muster als Vorbedingung für dessen Anwendbarkeit definiert ist.

Fleck et al. [FTLW14] stellen einen Ansatz vor, der die Auswahl und Konfiguration von Mustern zur Optimierung von Cloud-Anwendungen automatisiert, um spezifizierte Ziele, wie beispielsweise eine bestimmte Antwortzeit einer Komponente, zu erreichen. In der vorliegenden Arbeit wird die Selektion der anzuwendenden Muster manuell durch den Anwendungsmanager durchgeführt. Eine diesbezügliche Automatisierung wird in dieser Arbeit nicht behandelt.

Scheibler [Sch10] stellt eine modellgetriebene Methode zur automatisierten Erstellung ausführbarer Integrationslösungen auf Basis der Integrationsmuster (engl. „*Enterprise Integration Patterns*“) von Hohpe und Woolf [HW04] vor. Dabei können Integrationsmuster modelliert und in ausführbare BPEL-Prozessmodelle transformiert werden. Eine ähnliche Form der Musterautomatisierung wird in

der vorliegenden Arbeit angewendet, indem deklarative Managementmodelle durch Anwendung von Mustern generiert und anschließend in ausführbare Managementpläne transformiert werden, welche im Rahmen der prototypischen Implementierung in BPEL realisiert sind.

Um die Anwendung von Cloud-Architekturmustern zu unterstützen, schlagen Fehling et al. [FLR+11] vor, Muster mit ausführbaren Artefakten zu annotieren, die zur Bereitstellung einer Anwendung eingesetzt werden können. Das Konzept entspricht den Solution Implementations und ist somit, wie in [Abschnitt 2.4.2.1](#) diskutiert, nicht für die Automatisierung von Managementmustern geeignet.

2.5 Zusammenfassung

Die vorangegangenen Abschnitte haben gezeigt, dass (i) Automatisierung, (ii) die Integration von konzeptionellem und technischem Managementwissen sowie (iii) die Integration unterschiedlicher Managementparadigmen und -technologien essenziell für ein effizientes Anwendungsmanagement sind. Insbesondere bei umfangreichen Managementaufgaben führt deren Durchführung ohne Automatisierung, expliziten Entscheidungshilfen und strukturiertem Wissensmanagement zu großen Problemen und Ineffizienzen [TFL09].

Durch Automatisierung des Musteransatzes in Kombination mit etablierten Managementparadigmen und Technologien wird in dieser Arbeit mit der PALMA-Methode ein integriertes Verfahren vorgestellt, um diese Anforderungen durch die Definition eines durchgängigen, automatisierbaren Vorgehensmodells zu lösen. Die generische Form der Automatisierung von Managementmustern, wie sie in der vorliegenden Arbeit vorgestellt wird, ist basierend auf den existierenden Ansätzen nicht möglich, da sie entweder (i) detaillierte technische Expertise bezüglich der Verfeinerung für individuelle Anwendungsfälle erfordern oder (ii) die resultierenden Umsetzungen an individuelle Anwendungen gekoppelt sind und vorab implementiert werden müssen. Das in dieser Arbeit vorgestellte Vorgehensmodell löst diese Probleme durch Kombination deklarativer und imperativer Managementkonzepte, wodurch Musterlösungen generisch und somit unabhängig von einzelnen Anwendungen automatisiert werden können.

KAPITEL 3

MUSTERBASIERTE METHODE ZUM MANAGEMENT VON ANWENDUNGEN

Dieses Kapitel stellt eine musterbasierte Methode zum Management von Anwendungen vor, welche im Folgenden als *PALMA-Methode*¹ bezeichnet wird. Die Methode stellt [Forschungsbeitrag 1](#) der vorliegenden Arbeit dar und deckt die Managementphase des Anwendungsbetriebs ab (vgl. [Abschnitt 2.2.1](#)).

Die PALMA-Methode beschreibt ein hybrides Vorgehensmodell, um Managementprobleme bezüglich laufender Anwendungen durch die Anwendung von Managementmustern zu lösen, z. B. um Anwendungen ohne Ausfallzeit in eine Cloud-Umgebung zu migrieren. Die Methode kombiniert dazu den deklarativen Managementansatz, um die abstrakt beschriebenen Musterlösungen deklarativ und auf ähnlich hoher Abstraktionsebene zu modellieren, mit dem imperativen Ansatz, der die automatisierte Ausführung der modellierten Lösung ermöglicht.

¹Pattern-based Application Management

Das Kapitel ist folgendermaßen aufgebaut: Im nächsten Abschnitt wird die zentrale Rolle des Anwendungsmanagers eingeführt, welcher die vorgestellte PALMA-Methode anwendet. [Abschnitt 3.2](#) fasst Anforderungen an die Methode zusammen, die aus den Erkenntnissen der Analysen existierender Ansätze aus [Kapitel 2](#) abgeleitet werden. In [Abschnitt 3.3](#) wird die Methode im Detail vorgestellt. [Abschnitt 3.4](#) beschreibt eine Variante der Methode zur kombinierten Anwendung mehrerer Managementmuster. [Abschnitt 3.5](#) stellt eine Übersicht dar, wie die Methode und deren Variante durch die weiteren Forschungsbeiträge der vorliegenden Arbeit automatisiert werden können. Die Methode und deren Automatisierung werden in [Abschnitt 3.6](#) diskutiert.

3.1 Die Rolle des Anwendungsmanagers

In diesem Abschnitt wird die Rolle des *Anwendungsmanagers* definiert, welcher die zentrale handelnde Person der PALMA-Methode darstellt. Die Anwendungsmanager eines Unternehmens sind für den Betrieb der Geschäftsanwendungen verantwortlich und führen Aufgaben zu deren (i) Bereitstellung, (ii) Management und (iii) Außerbetriebnahme aus. Die Rolle grenzt sich von existierenden Rollen dahingehend ab, dass ein Anwendungsmanager durchzuführende Managementaufgaben nicht direkt ausführt, sondern ein formales Modell erstellt, welches die auszuführenden Aufgaben beschreibt. Auf Basis des Modells werden die modellierten Aufgaben anschließend vollautomatisiert, semi-automatisiert oder manuell durchgeführt. Der Anwendungsmanager ist für die Korrektheit der Durchführung verantwortlich und überwacht diese.

Die Rolle kann je nach Organisation von etablierten Rollen eingenommen werden und stellt somit eine Spezialisierung existierender Rollen dar. In traditionell geführten Unternehmen ist der Anwendungsmanager beispielsweise der Systemadministrator, in modernen DevOps-basierten Entwicklungsstrukturen [[Tur14](#)] kann der Softwareentwickler die Rolle einnehmen, um entwickelte Anwendungen bereitzustellen und zu betreiben. In dieser Arbeit wendet der Anwendungsmanager die PALMA-Methode an, um Anwendungen zu managen.

3.2 Anforderungen an eine Managementmethode

Dieser Abschnitt definiert generelle Anforderungen an eine Methode zum holistischen Management komplexer Geschäftsanwendungen aus der Sicht des Anwendungsmanagers, die aus den Erkenntnissen und Diskussionen des vorherigen Kapitels abgeleitet wurden. Jede Anforderung wird durch eine eindeutige Nummer identifiziert, die im Folgenden dazu verwendet wird, deren Umsetzung durch die PALMA-Methode zu verifizieren.

Anforderung 1 (Deklarative und technologieunabhängige Modellierung beider Aufgabentypen). Das Vorgehensmodell muss explizit die deklarative und technologieunabhängige Modellierung und Ausführung zustandsmodifizierender sowie zustandserhaltender Managementaufgaben unterstützen. Dadurch wird ein umfassendes Anwendungsmanagement ermöglicht sowie die komplexe und ineffiziente manuelle Erstellung und Adaption imperativer Managementpläne vermieden (abgeleitet aus [Abschnitt 2.3.1](#) und [Abschnitt 2.3.8](#)).

Anforderung 2 (Technologieunabhängige Anwendbarkeit). Aufgrund der zunehmenden Komplexität moderner Anwendungen muss das Vorgehensmodell grundsätzlich unabhängig von individuellen Technologien auf beliebige Anwendungen angewendet werden können. Ziel ist es, das Management komplexer komponierter Systeme zu ermöglichen, die aus heterogenen Komponenten bestehen und möglicherweise durch unterschiedliche Managementtechnologien verwaltet werden. Es darf nicht an spezifische Anbieter, Komponenten oder Managementtechnologien gekoppelt sein und muss diesbezüglich flexibel erweitert werden können (abgeleitet aus [Abschnitt 2.2.3](#)).

Anforderung 3 (Automatisierbarkeit). Um Anwendungsmanagement effektiv und fehlerfrei umsetzen zu können, ist die Automatisierbarkeit des Vorgehensmodells eine Kernanforderung (abgeleitet aus [Abschnitt 2.2.3.2](#)).

Anforderung 4 (Kontextsensitive Aufgabenanalyse). Um eine fehlerfreie Managementdurchführung zu gewährleisten, müssen Managementaufgaben im Kontext, in dem sie ausgeführt werden, vom Anwendungsmanager analysiert und gegebenenfalls adaptiert werden können (abgeleitet aus [Abschnitt 2.3.7](#)).

Anforderung 5 (Einflussnahme auf die Managementdurchführung). Das Vorgehensmodell muss es dem Anwendungsmanager erlauben die Managementdurchführung sowohl technisch als auch konzeptionell beeinflussen zu können, da dieser für die Ausführung verantwortlich ist (abgeleitet aus [Abschnitt 2.2.3.2](#)).

Anforderung 6 (Verlässliche, robuste und atomare Managementdurchführung). Das Vorgehensmodell muss eine verlässliche, robuste und atomare Managementdurchführung unterstützen: Fehler bei der Ausführung dürfen weder zum Ausfall der Anwendung noch zu inkonsistenten Anwendungszuständen führen. Scheitert die Managementdurchführung, müssen alle getätigten Änderungen rückgängig gemacht werden können (abgeleitet aus [Abschnitt 2.3.8](#)).

Anforderung 7 (Expertenkooperation). Da die erfolgreiche Durchführung von Managementaufgaben maßgeblich durch zwei verschiedene Faktoren in Form der konzeptionellen und der technischen Komplexität (siehe [Abschnitt 2.2.3](#)) beeinflusst wird, muss das Vorgehensmodell die Kooperation unterschiedlicher Experten, welche jeweils die Rolle des Anwendungsmanagers einnehmen, mit verschiedenen Verantwortlichkeiten, Fähigkeiten und Zuständigkeiten ermöglichen (engl. „*separation of concerns*“, abgeleitet aus [Abschnitt 2.2.4](#)).

Anforderung 8 (Anpassung an aktuelle Anwendungszustände). Aufgrund der Alterung von Prozessmodellen müssen diese automatisiert für aktuelle Anwendungsstrukturen und Zustände adaptiert bzw. generiert werden können. Eine Vorabimplementierung ist nicht effizient und gefährdet die erfolgreiche Ausführung der Managementaufgaben (abgeleitet aus [Abschnitt 2.3.5.3](#)).

Anforderung 9 (Reproduzierbarkeit und Nachvollziehbarkeit). Die Reproduzierbarkeit und Nachvollziehbarkeit von Managementdurchführungen spielen beim Anwendungsmanagement eine signifikante Rolle, da sie die Grundlage zur Vermeidung, Erkennung und Behebung von Fehlern darstellen sowie zur Überprüfung und Optimierung der Managementprozesse beitragen (abgeleitet aus [Abschnitt 2.3.4.3](#) und [Abschnitt 2.3.8](#)).

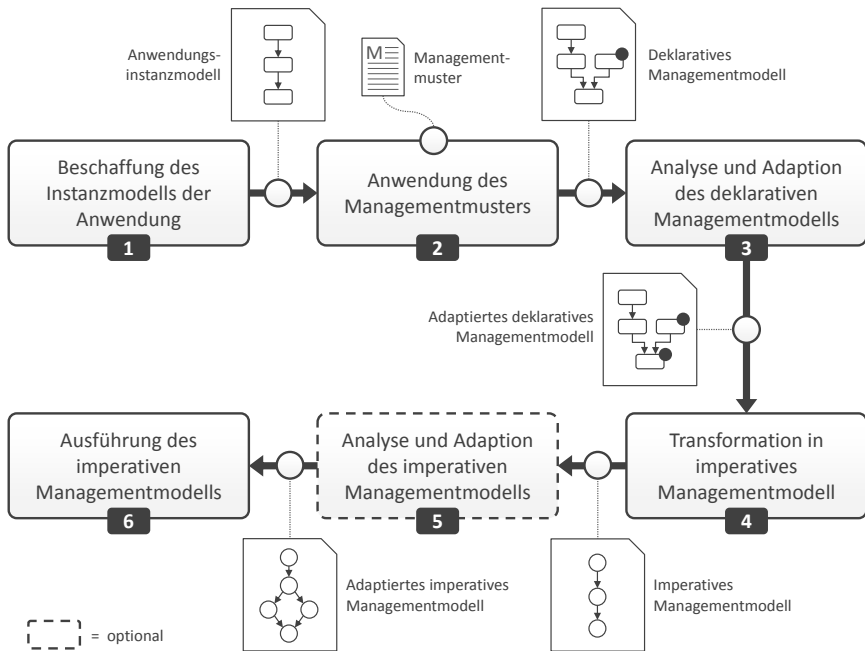


Abbildung 3.1: Schritte und involvierte Artefakte der PALMA-Methode

3.3 PALMA-Methode

Die PALMA-Methode definiert ein hybrides Vorgehensmodell zum musterbasierten Management laufender Anwendungen. Sie integriert dazu den deklarativen mit dem imperativen Managementansatz und wird grundsätzlich von Personen angewendet, welche die Rolle des Anwendungsmanagers (siehe [Abschnitt 3.1](#)) einnehmen und dadurch für das Management der betrieblichen Anwendungen zuständig sind. Die Methode ermöglicht es Anwendungsmanagern, das Management von Anwendungen durch Einsatz bewährter Managementmuster auf hoher deklarativer Abstraktionsebene zu modellieren und die Gesamtkosten sowie die Komplexität und Fehleranfälligkeit der Managementdurchführung zu reduzieren. Eine Übersicht der Methode sowie die jeweiligen Eingaben und Ausgaben der einzelnen Methodenschritte wird in [Abbildung 3.1](#) dargestellt.

3.3.1 Ziel und Ausführung der Methode

Der Anwendungsmanager führt die PALMA-Methode aus, um ein bestimmtes Managementproblem durch Anwendung eines Managementmusters zu lösen, z. B. um eine Anwendungskomponente ohne Ausfallzeit in eine Cloud-Umgebung zu migrieren (vgl. *Stateless Component Swapping-Muster* [FLR+13]). Vor Ausführung der Methode selektiert der Anwendungsmanager daher (i) die betroffene Anwendung sowie (ii) das anzuwendende Managementmuster.

Den Ausgangspunkt der Methode stellt das Instanzmodell der betroffenen Anwendung dar. Nach dessen Beschaffung (Schritt 1) wird das Instanzmodell durch Anwendung des ausgewählten Managementmusters zu einem deklarativen Managementmodell erweitert (Schritt 2), welches die auszuführenden Managementaufgaben in deklarativer Form spezifiziert. Nach dessen Verfeinerung und Analyse auf Korrektheit (Schritt 3) wird das deklarative Modell in ein imperatives Managementmodell transformiert (Schritt 4), das anschließend verifiziert (Schritt 5) und ausgeführt wird (Schritt 6). Der Anwendungsmanager durchläuft diese Schritte sukzessiv, kann jedoch zu jedem Zeitpunkt in einen beliebigen bereits abgeschlossenen Schritt zurückkehren und die Methode von diesem mit modifizierten Eingaben fortsetzen ([Anforderung 5](#)). Im Folgenden werden die Schritte sowie die involvierten Modelle im Detail beschrieben.

3.3.2 Schritt 1: Beschaffung des Instanzmodells der Anwendung

Ein aktuelles Instanzmodell der Anwendung, welches deren Struktur und gegenwärtigen Zustand beschreibt, wird erstellt oder abgerufen.

Im ersten Schritt wird ein aktuelles Instanzmodell der Anwendung beschafft, welches die Ausgangslage der nachfolgenden Schritte darstellt. Ein Instanzmodell dokumentiert sowohl (i) die gegenwärtige Struktur der Anwendung sowie (ii) Laufzeitinformationen, welche den aktuellen Zustand der Anwendung beschreiben (vgl. [Abschnitt 2.1.3](#)). Es beschreibt somit den vollständigen *Kontext*, in dem Aufgaben zum Management der Anwendung ausgeführt werden.

Die Beschaffung des Instanzmodells kann auf zwei Arten durchgeführt werden: Zum einen (i) setzen viele Unternehmen Konfigurationsmanagementdatenbanken (CMDB) zur Verwaltung von Anwendungskonfigurationen ein, welche aktuelle Instanzinformationen über laufende Anwendungen kontinuierlich automatisiert erfassen und aktualisieren (vgl. [Abschnitt 2.1.3](#)). Ist eine solche Datenbasis nicht gegeben, können Instanzmodelle auch (ii) manuell oder automatisiert auf Anfrage erstellt werden. Binz [[BBKL13a](#); [Bin15](#)] diskutiert hierzu unterschiedliche Ansätze auf Basis einer Literaturrecherche und zeigt die Automatisierbarkeit der Erstellung von Instanzmodellen (vgl. [Abschnitt 2.1.3](#)).

Da das Instanzmodell die Basis für die nachfolgenden Schritte darstellt, muss eine fehlerfreie Verwendung des Modells ermöglicht und gewährleistet werden. Deshalb muss das Instanzmodell Qualitätseigenschaften hinsichtlich Aktualität, Vollständigkeit und Korrektheit erfüllen [[Bin15](#)]: (i) Um eine Anwendung verlässlich managen zu können, muss das Instanzmodell die Anwendung exakt zum Zeitpunkt der Durchführung repräsentieren. Andernfalls können veraltete Informationen eine erfolgreiche Ausführung der Managementaufgaben gefährden, beispielsweise wenn im Instanzmodell beschriebene Komponenten zum Ausführungszeitpunkt eine neuere Konfiguration haben, welche die Managementaufgaben, die von einer veralteten Konfiguration ausgehen, beeinflusst [[HK10b](#)]. Die (ii) Vollständigkeit des Instanzmodells ist von grundlegender Bedeutung, da unbekannte Abhängigkeiten und Relationen zwischen Komponenten sowie fehlende Eigenschaften bzw. Konfigurationsinformationen über Komponenten die korrekte Ausführung der Managementaufgaben gefährden. Insbesondere wenn bereits installierte Software, welche die Managementaufgaben beeinflusst, nicht im Instanzmodell abgebildet ist (vgl. [Abschnitt 2.3.7](#)). Das dritte Qualitätskriterium betrifft die (iii) Korrektheit des Instanzmodells. Sind zum Beispiel Sicherheitskonfigurationen falsch beschrieben, können fehlende Zugriffsberechtigungen die erfolgreiche Managementdurchführung gefährden.

Um eine optimale Grundlage zur deklarativen Modellierung von Managementaufgaben zu bieten, muss das Instanzmodell zudem in geeigneter Granularität vorliegen (*Verkürzungsmerkmal*, siehe [[LL07](#)]): Die betroffenen Komponenten und Relationen der Anwendung müssen in angemessener Detailtiefe beschrie-

ben sowie deren Abhängigkeiten zueinander explizit erfasst werden. Die Granularität von Instanzmodellen kann in einigen existierenden Ansätzen gesteuert bzw. konfiguriert werden, beispielsweise bei der automatisierten Erstellung von Enterprise Topologie Graphen [BBKL13a; Bin15]. Das Ergebnis dieses Schrittes ist ein aktuelles, korrektes und vollständiges Instanzmodell der Anwendung.

3.3.3 Schritt 2: Anwendung des Managementmusters

Das selektierte Managementmuster wird auf das Instanzmodell der Anwendung angewendet, um daraus ein deklaratives Managementmodell zu erstellen, welches die auszuführenden Managementaufgaben deklariert.

Im zweiten Schritt wird das ausgewählte Managementmuster zur Lösung des Managementproblems angewendet. Jedes Managementmuster definiert in generischer Form die zur Lösung des Problems auszuführenden Managementaufgaben und wird auf das Instanzmodell der Anwendung angewendet, um ein sogenanntes *deklaratives Managementmodell* zu erstellen. Ein deklaratives Managementmodell beschreibt auszuführende Managementaufgaben deklarativ auf hoher Abstraktionsebene in Abhängigkeit des Anwendungsinstanzmodells:

Definition 11 (Deklaratives Managementmodell - informell). Ein deklaratives Managementmodell beschreibt (i) deklarativ abstrakte Managementaufgaben, welche bezüglich der Komponenten und Relationen einer Anwendung auszuführen sind, (ii) den gegenwärtigen Zustand der Anwendung sowie (iii) den Zielzustand, in den die Anwendung überführt werden soll. ■

Deklarative Managementmodelle beschreiben bezüglich der modellierten Aufgaben ausschließlich *was* ausgeführt werden muss, spezifizieren jedoch weder konzeptionell noch technisch, *wie* die Ausführung der deklarierten Managementaufgaben im Detail umgesetzt werden soll. Deklarative Managementmodelle charakterisieren sich daher grundlegend durch die Abwesenheit technischer Ausführungsdetails und ermöglichen dadurch das Modellieren technologieun-

abhängiger Managementaufgaben: Wird die Installation einer bestimmten Komponente deklariert, beschreibt ein deklaratives Modell nur, *dass* die Komponente installiert werden soll, jedoch ohne konkrete technische Anweisungen oder Aktivitäten zu spezifizieren, *wie* die Installation durchzuführen ist. Aufgrund dieser fehlenden technischen Ausführungsdetails sind deklarative Managementmodelle nicht direkt ausführbar und werden daher in den nachfolgenden Schritten in ausführbare imperative Managementmodelle transformiert.

Gegenüber den existierenden deklarativen Managementansätzen (vgl. [Abschnitt 2.3.3](#)) werden neben der *deskriptiven* Beschreibung der aktuellen Anwendungsinstanz und der *präskriptiven* Beschreibung des Zielzustandes in deklarativen Managementmodellen explizit die auszuführenden Managementaufgaben deklariert. Dadurch können sowohl zustandsmodifizierende als auch zustandserhaltende Managementaufgaben modelliert werden ([Anforderung 1](#)). Da deklarative Managementmodelle auf dem aktuellen Instanzmodell der Anwendung basieren, beschreiben diese inhärent den vollständigen Kontext, in dem die modellierten Managementaufgaben ausgeführt werden ([Anforderung 4](#)).

Deklarative Managementmodelle spezifizieren prinzipiell keine Ausführungsreihenfolge der Managementaufgaben. Diese wird bei der Transformation in ein imperatives Managementmodell individuell aus eindeutigen Abhängigkeiten zwischen den Managementaufgaben abgeleitet. Die Ausführungsreihenfolge zweier Aufgaben, d. h. deren partielle Ordnung bezüglich ihrer Ausführung, kann jedoch explizit deklariert werden, falls diese nicht eindeutig aus Abhängigkeiten ableitbar ist, jedoch zur Sicherstellung der gewünschten Managementsemantik erforderlich ist. Sind beispielsweise zwei unabhängige Managementaufgaben modelliert, welche Daten in eine Datenbank (i) importieren und (ii) exportieren, muss die beabsichtigte Reihenfolge explizit spezifiziert werden, da diese aufgrund des Fehlens direkter Abhängigkeiten nicht eindeutig bestimmt werden kann, aber das Resultat dennoch maßgeblich beeinflusst. Das Ergebnis dieses Schrittes ist ein deklaratives Managementmodell, welches die gegenwärtige Anwendungsinstanz beschreibt, die auszuführenden Managementaufgaben deklariert sowie den Zielzustand der Anwendung spezifiziert.

3.3.4 Schritt 3: Analyse und Adaption des deklarativen Managementmodells

Das deklarative Managementmodell wird analysiert und bezüglich
(i) nötiger Verfeinerungen und (ii) konzeptioneller Probleme adaptiert.

In diesem Schritt wird das deklarative Managementmodell analysiert und zwei Arten möglicher Adaptionen durchgeführt: (i) Verfeinerung des deklarativen Managementmodells und (ii) Beseitigung enthaltener Managementfehler.

Da Managementmuster eine Lösung generisch anwendbar beschreiben, muss das resultierende deklarative Managementmodell, je nach Typ und Abstraktionsgrad des angewandten Musters, gegebenenfalls nachträglich verfeinert werden. Wird beispielsweise ein Muster zur Migration einer Komponente in eine Cloud-Umgebung angewendet, muss die abstrakt beschriebene „Cloud“ durch Angabe eines konkreten Anbieters detaillierter spezifiziert werden. Eine Verfeinerung kann auch für deklarierte Managementaufgaben erforderlich sein, z. B. um Zugangsdaten für den gewählten Cloud-Anbieter anzugeben.

Im Gegensatz zu diesen bewusst erforderlichen Verfeinerungen können deklarative Managementmodelle auch Managementfehler enthalten, welche in diesem Schritt von Experten analysiert und behoben werden. Speziell wenn Anwendungen auf mehrere Rechenzentren verteilt sind oder aus heterogenen Anwendungskomponenten bestehen, können dem Anwendungsmanager unbekannte Abhängigkeiten zwischen verschiedenen Komponenten den Erfolg der Managementdurchführung und somit die Verfügbarkeit der gesamten Anwendung gefährden [BBK+14b]. Beispielsweise, wenn die Standard-Sicherheitskonfiguration des gewählten Cloud-Anbieters den geforderten externen Zugriff auf die zu migrierende Komponente verhindert und die Sicherheitskonfiguration nicht durch eine modellierte Managementaufgabe angepasst wird. Häufig sind Fehler nicht offensichtlich und schwer zu erkennen, insbesondere bei der Kombination mehrerer proprietärer Technologien. Daher wird das Modell von Experten auf Probleme und Fehler untersucht und gegebenenfalls korrigiert. Das Ergebnis ist ein korrektes deklaratives Managementmodell.

3.3.5 Schritt 4: Transformation in ein imperatives Managementmodell

Das deklarative Managementmodell wird in ein ausführbares imperatives Managementmodell überführt, welches die technische Durchführung der spezifizierten Managementaufgaben im Detail beschreibt.

Deklarative Managementmodelle beschreiben die auszuführenden Managementaufgaben ausschließlich abstrakt, d. h. sie spezifizieren nur *was* ausgeführt werden soll, jedoch nicht *wie*. Da die ausführungsrelevanten und technischen Informationen in diesen Modellen nicht enthalten sind, können deklarative Managementmodelle nicht direkt ausgeführt werden und müssen interpretiert oder weiterverarbeitet werden. Im vierten Schritt wird das deklarative Modell daher in ein *imperatives Managementmodell* transformiert, welches durch die Orchestrierung konkreter technischer Managementoperationen und der expliziten Modellierung des Kontroll- und Datenflusses zwischen den Operationsaufrufen zu einem ausführbaren Modell reift. Imperative Managementmodelle können mit den in [Abschnitt 2.3.4](#) beschriebenen Technologien umgesetzt werden, beispielsweise als Workflow [LR00]. Da die Ausführung von Managementaufgaben während des laufenden Anwendungsbetriebs die Verfügbarkeit der Anwendung direkt gefährdet, müssen zudem Fehlerbehandlungs- und Kompensationsmechanismen im imperativen Managementmodell beschrieben werden, welche die vollständige Wiederherstellung des ursprünglichen Anwendungszustands im Falle eines auftretenden, unbehebaren Fehlers ermöglichen. Imperative Managementmodelle werden wie folgt informell definiert:

Definition 12 (Imperatives Managementmodell - informell). Ein imperatives Managementmodell beschreibt eine ausführbare Orchestrierung technischer Managementoperationen, welche sowohl (i) den Kontrollfluss sowie (ii) den Datenfluss zwischen den Operationsaufrufen spezifiziert und im Fehlerfall automatisiert rückgängig gemacht bzw. kompensiert werden kann. ■

Bei dieser Transformation müssen somit die abstrakten, deklarativ modellierten Managementaufgaben in technische, imperative Anweisungen bzw. Aktivitäten übersetzt werden, welche die Aufgaben ausführen. Dazu können beispielsweise Skripte ausgeführt oder Managementdienste aufgerufen werden. Das Ergebnis dieses Schrittes ist ein ausführbares imperatives Managementmodell.

3.3.6 Schritt 5: Analyse und Adaption des imperativen Managementmodells

Das imperative Managementmodell wird bezüglich technischer Probleme analysiert und gegebenenfalls korrigiert. Dieser Schritt ist optional.

In diesem Schritt wird das imperative Managementmodell analysiert, um (i) Details der Transformation zu überprüfen und gegebenenfalls zu adaptieren oder um (ii) zusätzliche Managementaufgaben und Aktivitäten einzufügen.

Da deklarative Managementmodelle die Ausführungsreihenfolge der modellierten Managementaufgaben nur in Sonderfällen explizit spezifizieren (vgl. Schritt 2), werden diese Informationen typischerweise erst in Schritt 4, bei der Überführung in ein imperatives Managementmodell, generiert. Daher muss das resultierende imperative Modell möglicherweise hinsichtlich dieser Reihenfolgeaspekte untersucht werden, etwa um die Einhaltung der Firmen-Compliance zu garantieren oder andere Richtlinien bzw. Gesetze zu berücksichtigen.

Ein weiterer Zweck dieses Schrittes ist die Erweiterung bzw. Modifikation des imperativen Modells hinsichtlich Aktivitäten, welche deklarativ aufgrund der Komplexität nicht beschrieben werden können oder das Anwendungsmanagement nicht betreffen. Typische Beispiele für Letzteres sind Überwachungsaktivitäten, welche das Monitoring der Ausführung ermöglichen ohne einen funktionalen Einfluss auf die eigentlichen Managementaufgaben zu haben.

Da das imperative Managementmodell aus Schritt 4 bereits ausführbar ist, wird sowohl die Analyse als auch eine mögliche Adaption nicht zwangsweise benötigt. Daher ist dieser Schritt als optional anzusehen. Das Ergebnis dieses Schrittes ist ein gegebenenfalls adaptiertes imperatives Managementmodell.

3.3.7 Schritt 6: Ausführung des imperativen Managementmodells

Das imperative Managementmodell wird auf eine Ausführungsumgebung ausgeliefert, parametrisiert und ausgeführt.

Das finale imperative Managementmodell wird im letzten Schritt auf eine Ausführungsumgebung ausgeliefert, die Werte der Eingabeparameter spezifiziert und schlussendlich ausgeführt. Erst in diesem Schritt werden die Managementaufgaben real ausgeführt und die betroffene Anwendung beeinflusst. Während die vorherigen Schritte somit keinen realen Einfluss auf die Anwendung haben und zu jedem Zeitpunkt ohne Auswirkungen abgebrochen, modifiziert oder zurückgesetzt werden können, muss dieser Schritt entweder (i) vollständig erfolgreich durchgeführt oder (ii) im Fehlerfall unter Anwendung der im imperativen Managementmodell modellierten Fehlerbehandlungs- und Kompensationsmechanismen in seiner Gesamtheit zurückgesetzt werden.

Die Ausführung zustandsmodifizierender Managementaufgaben durch imperative Managementmodelle beeinflusst Zustände sowie Konfigurationen von Komponenten und Relationen der Anwendung und somit auch deren Instanzmodell (vgl. [Abschnitt 2.3.8](#)). Diese Eigenschaftsänderungen an Komponenten und Relationen können direkt vom imperativen Modell in Form einer Aktualisierung des zugehörigen Anwendungsinstanzmodells dokumentiert werden. Wird die Methode mehrfach ausgeführt, wird die Beschaffung des Instanzmodells in Schritt 1 durch diese direkte Aktualisierung vereinfacht und kann die aufwändige Neuerstellung ersetzen. Wird eine Konfigurationsmanagementdatenbank eingesetzt, um Instanzinformationen und Konfigurationen zu verwalten, muss das imperative Managementmodell diese Datenbank gegebenenfalls über die getätigten Änderungen informieren. Das Ergebnis dieses Schrittes ist die erfolgreiche Ausführung der Aufgaben für die reale Anwendung, falls keine Fehler aufgetreten sind. Andernfalls ist das Ergebnis der wiederhergestellte Ursprungszustand der Anwendung vor Ausführung der PALMA-Methode.

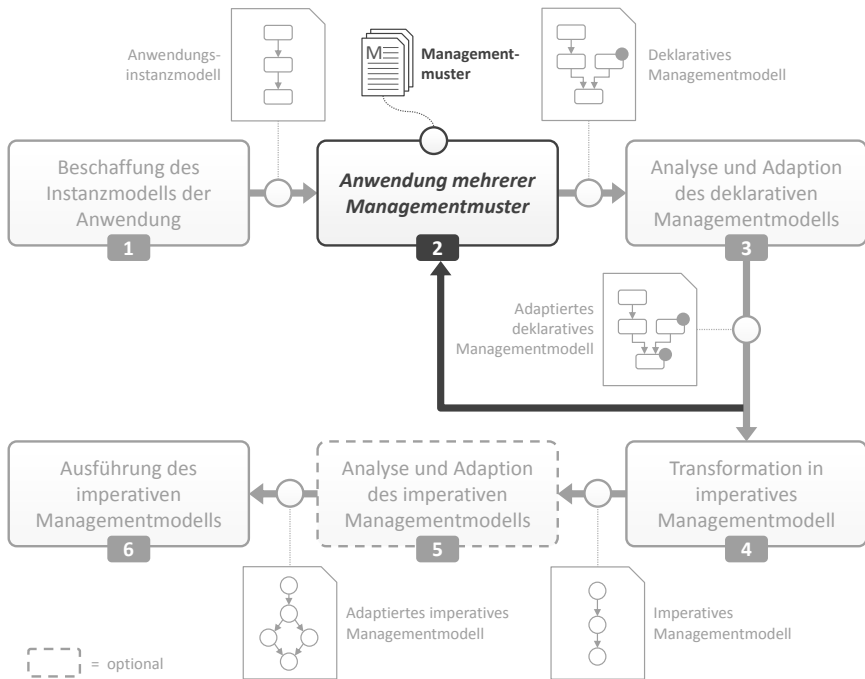


Abbildung 3.2: Variante der PALMA-Methode zur kombinierten und gemeinsamen Anwendung mehrerer Managementmuster

3.4 Variante zur Kombination mehrerer Managementmuster

Die PALMA-Methode unterstützt die *iterative Anwendung* mehrerer Managementmuster durch die separate Ausführung der Methode für jedes einzelne Muster. Diese Vorgehensweise ermöglicht jedoch nicht die kombinierte Anwendung mehrerer Muster innerhalb eines einzelnen Managementprozesses. Daher wird in diesem Kapitel eine Variante der PALMA-Methode beschrieben, durch die mehrere Managementmuster kombiniert auf eine Anwendung angewendet werden können. Eine Übersicht dieser Variante auf Basis der originalen Schritte sowie der involvierten Modelle wird in [Abbildung 3.2](#) dargestellt.

Die Änderungen am originalen Vorgehensmodell zur Umsetzung dieser Variante betreffen ausschließlich die Schritte 2 und 3. Anstatt in Schritt 2 nur ein einziges Managementmuster anzuwenden, können in dieser Variante gleichzeitig mehrere Muster kombiniert werden. Werden mehrere Managementmuster auf ein Instanzmodell angewendet, kann jedoch eine diesbezügliche Verfeinerung des resultierenden deklarativen Managementmodells in Schritt 3 nötig sein, um (i) die Managementaufgaben, welche von den Mustern deklariert werden, durch hinzufügen weiterer Aufgaben logisch miteinander zu verbinden oder (ii) die Ausführungsreihenfolge der Musterlösungen zu spezifizieren.

Die Variante unterstützt zudem die iterative Anwendung mehrerer Muster innerhalb der Methode. Dazu kann nach der Analyse in Schritt 3 wieder Schritt 2 ausgeführt werden, um weitere Muster anzuwenden. Durch diese beiden Möglichkeiten können mehrere Managementmuster kombiniert und die resultierenden deklarativen Managementmodelle flexibel adaptiert werden. Während die originale Methode durch sukzessive Anwendung mehrerer Managementmuster mehrfach hintereinander ausgeführt werden muss und die verschiedenen Experten mehrfach involviert werden, wird dies durch die Variante optimiert, indem ausschließlich die Schritte 2 und 3 iteriert werden. Dadurch werden Experten nur einmal für die anderen Methodenschritte involviert.

3.5 Anwendung und Automatisierung der PALMA-Methode

In diesem Abschnitt wird die (i) manuelle und (ii) automatisierte Anwendung der PALMA-Methode sowie die damit verbundenen Vor- und Nachteile diskutiert.

3.5.1 Manuelle Durchführung

Die PALMA-Methode kann von Anwendungsmanagern vollständig manuell durchgeführt werden. Die Differenzierung konzeptionellen und technischen Wissens und dessen strikte Aufteilung in separate Modelltypen, welche jedoch ineinander überführbar sind, ermöglicht die nahtlose Kooperation unterschiedlicher Experten mit verschiedenen Fähigkeiten, Zuständigkeiten und Erfahrungen. Die einzelnen Schritte können dadurch von unterschiedlichen Personen

durchgeführt werden, welche diesbezüglich die benötigte konzeptionelle oder technische Expertise vorweisen (engl. „*separation of concerns*“, [Anforderung 7](#)). Der Einsatz von Managementmustern unterstützt die Methodenanwender dabei, die auszuführenden Managementaufgaben deklarativ korrekt auf Basis bewährten und erprobten Wissens zu spezifizieren. Eine vollständig manuelle Durchführung ist jedoch aufgrund zeitintensiver Arbeiten, wie der Erstellung des Instanzmodells, sehr aufwändig und führt zu hohen Personalkosten. Zudem gefährden manuelle Fehler bei der Umsetzung komplexer Aufgaben, wie etwa der Orchestrierung von Managementtechnologien in Schritt 4, sowohl die Effektivität als auch die Effizienz der Durchführung (vgl. [Abschnitt 2.2.3.1](#)). Speziell der Einsatz existierender Arbeiten und Technologien für die Erstellung von Instanzmodellen (vgl. [Abschnitt 2.1.3](#)) und der Ausführung imperativer Managementmodelle, z. B. durch die Workflow-Technologie (vgl. [Abschnitt 2.3.4](#)), können durch Automatisierung der jeweiligen Schritte die Kosten, Fehleranfälligkeit und Durchführungszeit signifikant senken. Eine vollständig manuelle Anwendungsform ist folglich nicht empfehlenswert.

Eine semi-automatisierte Anwendungsform kann durch Nutzung existierender Technologien und der daraus resultierenden Kombination von manuellen und automatisierten Schritten deutlich zur Optimierung der Anwendung beitragen. Da in vielen Anwendungsgebieten, beispielsweise dem Cloud Computing und dem Management wichtiger Geschäftsanwendungen, eine schnelle und fehlerfreie Durchführung der Managementaufgaben gewährleistet werden muss, sind die oben genannten Probleme in diesen Bereichen, selbst die semi-automatisierte Variante, jedoch nicht akzeptabel. Um eine effiziente Durchführung zu ermöglichen, wird daher im folgenden Abschnitt beschrieben, wie alle Schritte der PALMA-Methode durch existierende Arbeiten und durch die weiteren Forschungsbeiträge der vorliegenden Arbeit vollständig automatisiert werden können. Hauptziel dabei ist es, die Gesamtkosten, die Ausführungszeit und die Fehleranfälligkeit des Managementvorgangs zu reduzieren.

Um die Anwendung von Managementmustern in Schritt 2 zu automatisieren, stellt [Forschungsbeitrag 2](#) die deklarative Sprache DMMN zur Modellierung von Managementaufgaben vor. Die Sprache ermöglicht die Erstellung deklarativer Managementmodelle gemäß [Definition 11](#). [Forschungsbeitrag 4](#) beschreibt ein Konzept, um die Anwendung von Managementmustern auf Basis von DMMN zu automatisieren, indem DMMN-Modelle, welche die Musterlösung deklarativ für individuelle Anwendungen spezifizieren, automatisiert generiert werden.

Die Analyse und Adaption deklarativer Managementmodelle in Schritt 3 wird durch [Forschungsbeitrag 5](#) automatisiert, der es ermöglicht, Managementaufgaben in deklarativen Managementmodellen automatisiert auf Fehler zu analysieren und diese wenn möglich direkt zu korrigieren.

Die Transformation deklarativer Managementmodelle in ausführbare imperative Managementmodelle wird durch [Forschungsbeitrag 3](#) ermöglicht. Dabei werden wiederverwendbare Managementbausteine in Form von Management-Planlets orchestriert, um die in deklarativen Managementmodellen abstrakt spezifizierten Managementaufgaben auszuführen.

Zur Modellierung und Ausführung imperativer Managementmodelle werden in der vorliegenden Arbeit etablierte Workflow-Technologien [[LR00](#)] eingesetzt. Imperative Managementmodelle werden mittels der Workflow-Sprache BPEL realisiert [[OAS07b](#)], wodurch diese von standardkonformen Workflow-Maschinen ausgeführt werden können. Beispielsweise durch den *WSO2 Business Process Server 2.0*¹, der auch bei der prototypischen Realisierung des PALMA-Frameworks im Rahmen von [Forschungsbeitrag 6](#) eingesetzt wird.

Die automatisierte Analyse und Adaption dieser Workflows in Schritt 5 kann durch existierende Arbeiten, wie beispielsweise von Schleicher [[Sch14](#)] vorgestellt, umgesetzt werden und ist daher nicht Teil dieser Dissertation.

¹<http://wso2.com/products/business-process-server/>

3.6 Diskussion und Umsetzung der erhobenen Anforderungen

Die PALMA-Methode stellt ein hybrides Vorgehensmodell dar, welches die Stärken des deklarativen und des imperativen Managementparadigmas vereint. Während beide Typen an Managementaufgaben (vgl. [Abschnitt 2.3.1](#)) deklarativ und auf hoher Abstraktionsebene in Form von deklarativen Managementmodellen modelliert werden können ([Anforderung 1](#)), ermöglicht die automatisierte Transformation in imperative Modelle, speziell die eingesetzte Workflow-Technologie, deren robuste und verlässliche Ausführung ([Anforderung 6](#)) sowie die uneingeschränkte Beeinflussbarkeit der Managementdurchführung ([Anforderung 5](#)). Durch deklarative Managementmodelle können Managementaufgaben zudem unabhängig der letztendlich zur Umsetzung eingesetzten Managementtechnologien modelliert werden, da diese erst bei der Transformation in imperative Managementmodelle selektiert und integriert werden ([Anforderung 1](#)). Des Weiteren können die generierten Prozessmodelle reproduzierbar und nachvollziehbar ausgeführt werden, da Workflow-Sprachen wie BPEL eine explizite operationale Semantik definieren ([Anforderung 9](#)). Die Transformation abstrakter Managementanforderungen in imperative Prozesse entspricht der generellen Vorgehensweise des ITIL Change Managements [[Can07](#)], welches deklarative Requests for Change (RFCs) als Vorlage zur Erzeugung imperativer Change Plans einsetzt (siehe [Abschnitt 2.2.2](#)). Die PALMA-Methode könnte daher generell zur Umsetzung der ITIL-Konzepte eingesetzt werden, was jedoch nicht Gegenstand der vorliegenden Arbeit ist.

Durch die Aufteilung der Managementdurchführung in verschiedene Schritte, die nahtlos ineinander überführt werden, ermöglicht die Methode die Kollaboration unterschiedlicher Experten ([Anforderung 7](#)), wodurch deklarative Managementmodelle und die resultierenden Managementpläne zudem kontextsensitiv auf Probleme analysiert und gegebenenfalls adaptiert werden können ([Anforderung 4](#)). Fehlerquellen werden dadurch systematisch auf Basis bewährten Wissens und durch dessen Automatisierung reduziert.

Durch die Generierung imperativer Managementmodelle auf Grundlage eines aktuellen Instanzmodells spielt die Alterung von Managementprozessen keine

Rolle, da die generierten Pläne explizit für den aktuellen Anwendungszustand erstellt werden ([Anforderung 8](#)). Zudem werden konkrete Managementtechnologien erst während der Transformation in Schritt 4 mit einbezogen, wodurch die Methode selbst technologieunabhängig ist ([Anforderung 2](#)).

Die Automatisierung der Musteranwendung löst das Effizienzproblem, das bei deren manuellen Anwendung entsteht ([Anforderung 3](#)). Dadurch können selbst unerfahrene Administratoren komplexe Managementaufgaben durchführen. Zudem können Unternehmen internes Managementwissen in Form von Mustern erfassen, um Anwendungsmanagement firmenweit zu automatisieren und zu homogenisieren. Managementwissen kann durch dieses Konzept unabhängig von individuellen Anwendungsinstanzen und Personen erfasst, kommuniziert und ausgeführt werden. Das Konzept der Managementmuster ermöglicht es Unternehmen zudem, konzeptionelle Managementexpertise von Dritten einzukaufen. In Kombination mit der Transformation in imperative Managementmodelle ermöglicht dieses Konzept die Automatisierung von Managementwissen sowohl auf konzeptioneller als auch auf technischer Ebene.

3.6.1 Abgrenzung zu verwandten Arbeiten

Boutaba und Znaty [[BZ95](#)] stellen eine generische Architektur für Managementsysteme vor, welche eine zur PALMA-Methode konzeptionell ähnliche Managementdurchführung ermöglicht. Das vorgestellte Konzept gruppiert technische Managementfunktionen individueller Domänen in Form von Subsystemen, welche auf Basis abstrahierter Schnittstellen hierarchisch strukturiert und rekursiv kombiniert werden können. Die Architektur ermöglicht dadurch die Beschreibung von Managementzielen auf hoher Abstraktionsebene, welche durch *Policies* konkretisiert und anschließend durch Nutzung von Vorlagen (*Plan Templates*) in ausführbare Managementpläne übersetzt werden. Ein Metamodell zur Beschreibung von Managementaufgaben sowie Details zu deren Übersetzung in ausführbare Pläne werden jedoch nicht vorgestellt und ausschließlich auf konzeptioneller Architekturebene diskutiert.

KAPITEL 4

DEKLARATIVE SPRACHE ZUR MODELLIERUNG VON MANAGEMENTAUFGABEN

Dieses Kapitel stellt die formale Modellierungssprache DMMN¹ vor, die zur Modellierung deklarativer Managementmodelle gemäß [Definition 11](#) eingesetzt werden kann. Die Sprache trägt zur Realisierung der PALMA-Methode ([Kapitel 3](#)) bei und stellt [Forschungsbeitrag 2](#) der vorliegenden Dissertation dar.

Das DMMN-Metamodell, welches in diesem Kapitel beschrieben wird, ermöglicht die deklarative Modellierung zustandserhaltender sowie zustandsmodifizierender Managementaufgaben in Abhängigkeit des Instanzmodells, welches die betroffene Anwendung repräsentiert. Dadurch können beliebige Aufgaben des Anwendungsmanagements spezifiziert werden (vgl. [Abschnitt 2.3.3.5](#)). Zusätzlich kann durch DMMN auch die Bereitstellung einer Anwendung sowie deren

¹Declarative Application Management Modelling and Notation

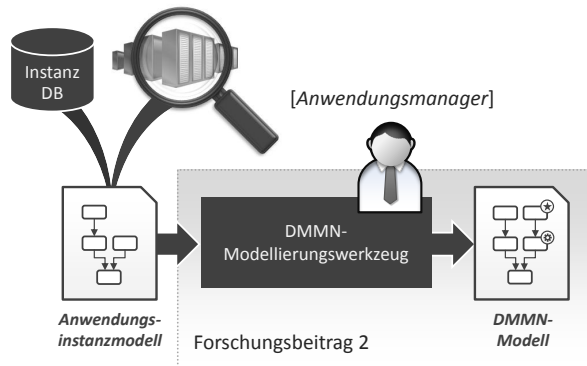


Abbildung 4.1: Konzept der DMMN-Modellierung

Außerbetriebnahme modelliert werden, wodurch die Sprache generell in allen Phasen des Anwendungsbetriebs eingesetzt werden kann (vgl. [Abschnitt 2.2.1](#)). In der vorliegenden Arbeit wird die Modellierungssprache DMMN bei der Umsetzung der PALMA-Methode eingesetzt, um die Lösung eines anzuwendenden Managementmusters automatisiert für eine bestimmte Anwendung in ein deklaratives Managementmodell zu übersetzen. Dafür wird in [Abschnitt 4.2](#) das DMMN-Metamodell definiert und dessen Konzepte erläutert.

Zur manuellen Modellierung von DMMN-Modellen sowie zur Verfeinerung angewendeter Managementmuster in Schritt 3 der PALMA-Methode wird in [Abschnitt 4.4](#) eine grafische Notation für das Metamodell eingeführt, welche die Basis zur Entwicklung eines grafischen DMMN-Modellierungswerkzeugs darstellt. Das Konzept dieses Werkzeugs ist in [Abbildung 4.1](#) beschrieben. Es ermöglicht dem Anwendungsmanager das Instanzmodell einer Anwendung als Basis zur Modellierung eines DMMN-Modells zu verwenden. Das Werkzeug wird durch die weiteren Beiträge dieser Arbeit schrittweise zum PALMA-Framework weiterentwickelt und um Konzepte zur automatisierten (i) Anwendung von Managementmustern, (ii) Plangenerierung und (iii) kontextsensitiven Analyse und Korrektur erweitert. Die Gesamtarchitektur sowie die prototypische Implementierung des PALMA-Frameworks werden detailliert in [Kapitel 8](#) vorgestellt.

4.1 Research Design der Entwicklung von DMMN

DMMN ist eine domänenspezifische Sprache (DSL) [Hud98; MHS05] zur Modellierung auszuführender Managementaufgaben. Gegenüber generisch einsetzbarer Sprachen (engl. „*general-purpose languages*“) und Technologien, wie beispielsweise Workflows, die zur Ausführung verschiedenster Formen von Geschäftsprozessen eingesetzt werden (vgl. [Abschnitt 2.3.4](#)), sind domänenspezifische Sprachen für ein bestimmtes Anwendungsgebiet optimiert. Durch Definition von optimal zugeschnittenen Konstrukten und Notationen ermöglichen DSLs eine ausdrucksstärkere und genauere Beschreibung der Ziele und vereinfachen dadurch die Nutzung der Sprache, wodurch eine höhere Produktivität und geringere Wartungskosten erzielt werden können [MHS05].

Das wissenschaftliche Vorgehen bezüglich der Entwicklung von DMMN ist auf dem von Mernik et al. [MHS05] vorgestellten Prozess begründet, der fünf Phasen zur systematischen DSL-Entwicklung definiert: (i) Entscheidung, (ii) Analyse, (iii) Entwurf, (iv) Implementierung und (v) Einsatz. Die Entscheidung zur Entwicklung einer neuen DSL basiert auf den Erkenntnissen der Analyse existierender Managementansätze und Modellierungssprachen in [Kapitel 2](#): DMMN basiert auf einigen Konzepten des TOSCA-Standards, modifiziert und erweitert diese jedoch zum Teil signifikant zur Steigerung der Flexibilität. Beispielsweise vereinigt DMMN die Modellformen der Anwendungstopologie mit dem Instanzmodell und ermöglicht die deklarative Spezifikation auszuführender Managementaufgaben. Um dies in TOSCA zu realisieren, wäre die Modifikation großer Teile der Syntax und Semantik nötig gewesen. Zudem greift DMMN nicht alle Konzepte von TOSCA auf, um die Komplexität der Sprache im Vergleich mit TOSCA zu senken. Beispielsweise entfällt die Definition von *Requirements* und *Capabilities* [OAS13b]. Aus diesen Gründen wurde die Neuentwicklung einer domänenspezifischen Modellierungssprache gewählt. In der zweiten Phase, dem Entwurf, wurde die Domäne des Anwendungsmanagements detailliert analysiert und Anforderungen an eine Modellierungssprache gesammelt. Diese Analyse ist im Rahmen des Grundlagenkapitels dokumentiert (siehe [Kapitel 2](#)), die daraus abgeleiteten Anforderungen, welche auch die umrahmende PALMA-

Methode betreffen, in [Abschnitt 3.2](#). Die erste und zweite Phase haben sich dabei gegenseitig beeinflusst: Die Entscheidung zur Entwicklung einer neuen DSL basiert auf den wissenschaftlichen Erkenntnissen der Analyse. Umgekehrt wurde nach der Entscheidung, eine neue DSL zu entwickeln, die Analyse hinsichtlich weiterer Anforderungen an Modellierungssprachen ausgerichtet. In der Entwurfsphase (engl. „*design*“) wurde das Metamodell von DMMN formal definiert und Konzepte aus existierenden Sprachen berücksichtigt. Dieses Vorgehen wird auch als „*Piggyback*“ bezeichnet [[MHS05](#)] und basiert ebenfalls auf den Erkenntnissen des Grundlagenkapitels 2. In der Implementierungsphase wurde untersucht, wie Modelle der entwickelten DSL ausgeführt werden können. Dies stellt [Forschungsbeitrag 3](#) der vorliegenden Arbeit dar. Die letzte Phase des Einsatzes der DSL (engl. „*deployment*“) wird in der vorliegenden Arbeit im Rahmen der Musterautomatisierung in [Forschungsbeitrag 4](#) realisiert.

4.2 DMMN-Metamodell

DMMN ist eine formale Sprache, welche sowohl Syntax als auch Semantik definiert. Das DMMN-Metamodell ermöglicht die deklarative Modellierung auszuführender Managementaufgaben in Abhängigkeit der Anwendungsstruktur, sowie die Abbildung aller verfügbaren technischen Managementoperationen der Anwendungskomponenten und deren Relationen. Die Klassen des DMMN-Metamodells sind in drei Schichten unterteilt: (i) Topologieschicht, (ii) technische Managementschicht und (iii) deklarative Managementschicht. Eine strukturelle Übersicht des Metamodells ist in [Abbildung 4.2](#) dargestellt. Abstrakte Klassen werden darin durch eine gepunktete Umrandung markiert.

Zur eindeutigen Unterscheidung zwischen einer realen Entität und deren Repräsentation im Modell wird dem Bezeichner der Repräsentation das Suffix „*element*“ angehängt. Folglich bezeichnet z. B. der Begriff „Komponente“ die reale Entität, „Komponentenelement“ deren Repräsentation im Modell. Wo nicht anders definiert sind Abbildungen linkstotale und rechtseindeutige Relationen. Die Prädikatenlogik erster Stufe dient der Formulierung mathematischer Ausdrücke, Σ ist die Menge der Unicode-Zeichen mit Ausnahme des Sterns (*).

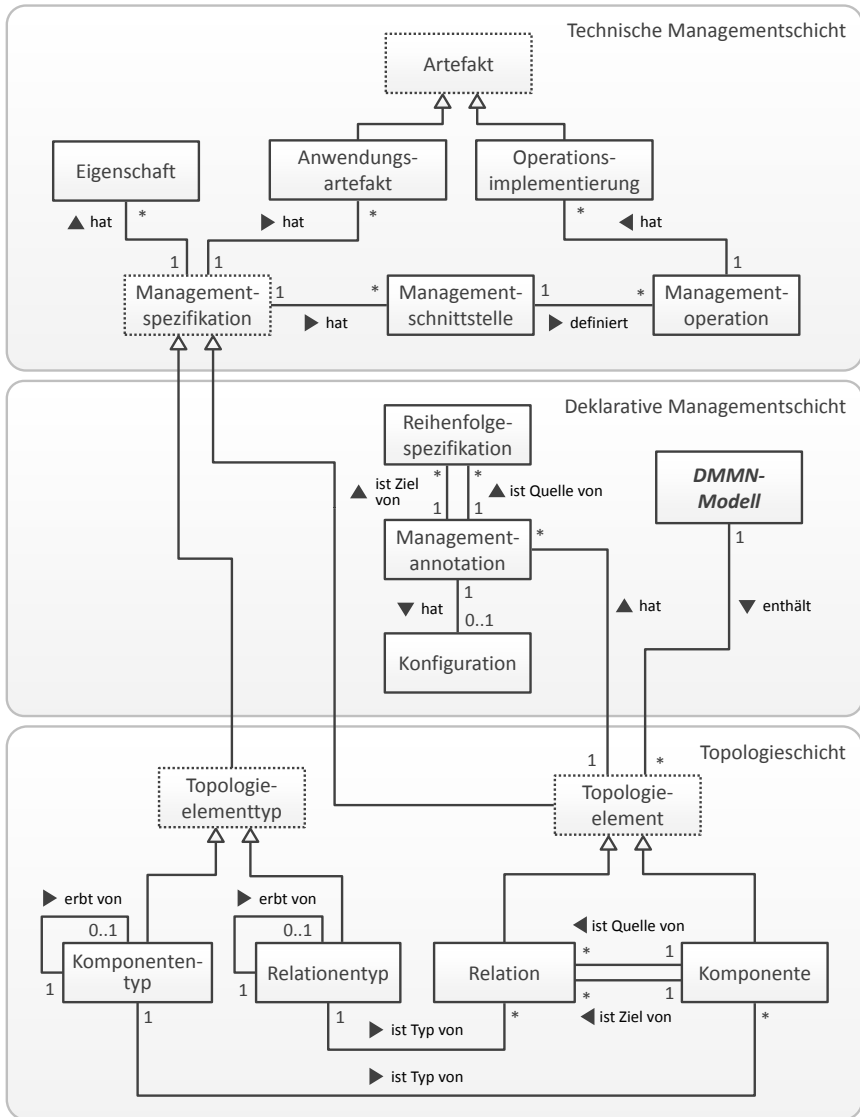


Abbildung 4.2: Schichten, Klassen und Relationen des DMMN-Metamodells

4.2.1 DMMN-Modell

Im Zentrum des DMMN-Metamodells steht die Klasse „DMMN-Modell“, welche folgendermaßen formal definiert wird:

Definition 13 (DMMN-Modell). Ein DMMN-Modell ist ein gerichteter, möglicherweise zyklischer Multigraph mit gefärbten Knoten und gewichteten Kanten. Sei D die Menge aller DMMN-Modelle, dann wird $d \in D$ definiert als 19-Tupel:

$$d = (K_d, R_d, KT_d, RT_d, E_d, I_d, O_d, A_d, MA_d, C_d, P_d, \\ typ_d, supertyp_d, eigeneEigenschaften_d, eigeneSchnittstellen_d, \\ operationen_d, eigeneArtefakte_d, annotationen_d, konfiguration_d)$$

Dabei ist:

- K_d die Menge aller Komponentenelemente in d ([Definition 15](#)),
- R_d die Menge aller Relationenelemente in d ([Definition 16](#)),
- KT_d die Menge aller Komponententypen in d ([Definition 19](#)),
- RT_d die Menge aller Relationentypen in d ([Definition 20](#)),
- E_d die Menge aller Eigenschaftenelemente in d ([Definition 26](#)),
- I_d die Menge aller Schnittstellenelemente in d ([Definition 32](#)),
- O_d die Menge aller Operationenelemente in d ([Definition 35](#)),
- A_d die Menge aller Artefaktelemente in d ([Definition 39](#)),
- MA_d die Menge aller Annotationenelemente in d ([Definition 43](#)),
- C_d die Menge aller Konfigurationenelemente in d ([Definition 46](#)),
- P_d die Menge aller Reihenfolgespezifikationenelemente in d ([Definition 49](#)),
- typ_d die Abbildung, welche jedem Komponentenelement und jedem Relationenelement in d dessen Typenlement zuordnet ([Definition 22](#)),
- $supertyp_d$ die Abbildung, welche jedem Komponenten- und Relationentypen in d dessen Supertypenlement zuordnet ([Definition 23](#)),

- *eigeneEigenschaften_d* die Abbildung, welche jedem Komponenten-, Relationen-, Komponententyp- und Relationentypenelement in *d* seine Eigenschaftenelemente zuordnet (Definition 27),
- *eigeneSchnittstellen_d* die Abbildung, welche jedem Komponenten-, Relationen-, Komponententyp- und Relationentypenelement in *d* seine Management-schnittstellenelemente zuordnet (Definition 33),
- *operationen_d* die Abbildung, welche jedem Schnittstellenelement in *d* seine Operationenelemente zuordnet (Definition 36),
- *eigeneArtefakte_d* die Abbildung, welche jedem Topologieelement und Operationenelement in *d* dessen Artefaktelemente zuordnet (Definition 40),
- *annotationen_d* die Abbildung, welche jedem Komponenten- und Relationenelement in *d* seine Annotationenelemente zuordnet, welche die auszuführenden Managementaufgaben beschreiben (Definition 44),
- *konfiguration_d* die Abbildung, welche jedem Annotationenelement sein Konfigurationenelement zuordnet (Definition 47).

Dabei darf kein repräsentierter Komponenten- oder Relationentyp direkt oder transitiv von sich selbst erben. ■

Ein DMMN-Modell beschreibt auszuführende Managementaufgaben bezüglich genau einer Anwendung. Wird in den folgenden Definitionen der Terminus „Anwendung“ verwendet, bezeichnet dieser somit genau diejenige Anwendung. Diese muss nicht zwingendermaßen schon in Betrieb sein, da DMMN auch zur deklarativen Modellierung deren Bereitstellung eingesetzt werden kann. Zum übersichtlicheren Umgang mit Tupeln wird die Projektion π_j verwendet:

Definition 14 (Projektion π_j). Seien $X_1, \dots, X_n$ beliebige Mengen, $n \in \mathbb{N}^+$, dann ist das kartesische Produkt $X = X_1 \times \dots \times X_n$ die Menge aller *n*-Tupel, deren *i*-te Komponente, $i \in \mathbb{N}^+$, $1 \leq i \leq n$, ein Element $x_i \in X_i$ ist. Sei $j \in \mathbb{N}^+$, $1 \leq j \leq n$, dann ist die Projektion π_j eine Abbildung, welche jedes dieser *n*-Tupel auf seine *j*-te Komponente abbildet:

$$\pi_j: X_1 \times \dots \times X_j \times \dots \times X_n \rightarrow X_j, (x_1, \dots, x_j, \dots, x_n) \mapsto x_j, x_j \in X_j \quad \blacksquare$$

4.2.2 Topologieschicht

Die Topologieschicht ermöglicht die Formalisierung der Struktur der betroffenen Anwendung, für welche Managementaufgaben ausgeführt werden sollen, in Form eines gerichteten, möglicherweise zyklischen Multigraphen.

4.2.2.1 Komponenten und Relationen

Die Knoten eines DMMN-Modells repräsentieren die Komponenten einer Anwendung. Eine Komponente ist dabei ein physikalisch, funktional oder logisch eigenständiger Bestandteil der Anwendung, beispielsweise eine Datenbank, ein Betriebssystem, ein Webserver oder eine virtuelle Maschine.

Definition 15 (Komponentenelemente). Sei $d \in D$ ein DMMN-Modell, dann enthält die endliche Menge $K_d := \pi_1(d) = \{k_1, k_2, \dots, k_n \mid n \in \mathbb{N}\}$ alle Instanzen k_i der DMMN-Klasse „Komponente“, welche jeweils eine Komponente der Anwendung repräsentiert. Diese Instanzen werden als *Komponentenelemente des Modells* bezeichnet. Jedes Komponentenelement $k_i \in K_d$ ist ein Paar:

$$K_d \subseteq \Sigma^+ \times \Sigma^+$$

$$k_i = (Id, Name)$$

Dabei ist:

- $\pi_1(k_i) = Id, Id \in \Sigma^+$, ein global eindeutiges, nicht-leeres Wort, das die repräsentierte Komponente eindeutig identifiziert,
- $\pi_2(k_i) = Name, Name \in \Sigma^*$, ein innerhalb K_d eindeutiges, nicht-leeres Wort, das den Namen der dargestellten Komponente repräsentiert.

■

Eine reale Komponente kann gleichzeitig in mehreren DMMN-Modellen durch Komponentenelemente repräsentiert werden, beispielsweise wenn verschiedene Anwendungen zum Betrieb denselben Server nutzen. Die repräsentierte Komponente wird durch das Attribut „Id“ modellübergreifend eindeutig identi-

fiziert. Ist der Wert dieses Attributs zweier Komponentenelemente identisch, repräsentieren beide Elemente somit dieselbe reale Entität. Innerhalb eines DMMN-Modells $d \in D$ gilt jedoch folgende Einschränkung:

$$\forall k_1, k_2 \in K_d: \pi_1(k_1) = \pi_1(k_2) \Rightarrow k_1 = k_2$$

Um eine Komponente im jeweiligen individuellen Anwendungskontext semantisch sinnvoll benennen zu können, spezifiziert jedes Komponentenelement zusätzlich das Attribut „Name“, welches individuell gewählt werden kann, jedoch innerhalb des Modells eindeutig sein muss. Dadurch kann eine Komponente durch unterschiedliche Namen in verschiedenen Modellen sinnvoll benannt werden, während die eindeutige Identifikation der Komponente durch das Attribut „Id“ des Komponentenelements gewährleistet wird. Innerhalb eines DMMN-Modells $d \in D$ gilt somit auch folgende Einschränkung:

$$\forall k_1, k_2 \in K_d: \pi_2(k_1) = \pi_2(k_2) \Rightarrow k_1 = k_2$$

Die Kanten eines DMMN-Modells repräsentieren die gerichteten physikalischen, funktionalen oder logischen Relationen zwischen genau zwei Komponenten der modellierten Anwendung. Zum Beispiel, dass zwei Server durch ein Netzwerk physikalisch miteinander verbunden sind.

Definition 16 (Relationenelemente). Sei $d \in D$ ein DMMN-Modell, dann enthält die endliche Menge $R_d := \pi_2(d) = \{r_1, r_2, \dots, r_n \mid n \in \mathbb{N}\}$ alle Instanzen r_i der DMMN-Klasse „Relation“, die jeweils eine Relation zwischen zwei Anwendungskomponenten repräsentiert. Diese Instanzen werden als *Relationenelemente des Modells* bezeichnet. Jedes Relationenelement $r_i \in R_d$ ist als Quadrupel definiert:

$$R_d \subseteq \Sigma^+ \times \Sigma^+ \times K_d \times K_d$$

$$r_i = (Id, Name, k_s, k_t)$$

Dabei ist:

- $\pi_1(r_i) = Id, Id \in \Sigma^+$, ein global eindeutiges, nicht-leeres Wort, das die repräsentierte Relation eindeutig identifiziert,
- $\pi_2(r_i) = Name, Name \in \Sigma^+$, ein innerhalb R_d eindeutiges, nicht-leeres Wort, das den Namen der dargestellten Relation repräsentiert,
- $\pi_3(r_i) = k_s, k_s \in K_d$, das Komponentenelement, das die Ursprungskomponente der beschriebenen Relation repräsentiert,
- $\pi_4(r_i) = k_t, k_t \in K_d$, das Komponentenelement, das die Zielkomponente der beschriebenen Relation repräsentiert.

Die Semantik und Einschränkungen der Attribute „ Id “ und „ $Name$ “ entsprechen den gleichnamigen Attributen der Komponentenelemente. ■

Die Komponenten- und Relationenelemente eines DMMN-Modells beschreiben somit die Topologie der repräsentierten Anwendung gemäß [Definition 2](#).

Definition 17 (Topologieelemente). Seien K_d die Komponentenelemente und R_d die Relationenelemente eines DMMN-Modells $d \in D$, dann wird deren Vereinigung $TE_d := K_d \cup R_d$ als *Topologieelemente des Modells* bezeichnet. ■

Die endliche Menge der Topologieelemente enthält damit alle Instanzen der abstrakten DMMN-Klasse „Topologieelement“ eines DMMN-Modells. Die endliche Menge aller Topologieelemente aller DMMN-Modelle wird definiert wie folgt:

Definition 18 (Menge aller Topologieelemente). Sei $D = \{d_1, \dots, d_n \mid n \in \mathbb{N}\}$ die Menge aller DMMN-Modelle, TE_{d_i} die Menge aller Topologieelemente eines DMMN-Modells $d_i \in D$, dann enthält die endliche Menge $TE := \{te_1, te_2, \dots, te_n \mid n \in \mathbb{N}\}$ alle Topologieelemente aller DMMN-Modelle $d_i \in D$. Diese wird als *Menge aller Topologieelemente* bezeichnet und folgendermaßen definiert:

$$TE := \bigcup_{d_i \in D} TE_{d_i}$$

■

4.2.2.2 Typen

Um die physische, funktionale oder logische Bedeutung der repräsentierten Komponenten und Relationen zu beschreiben, wird die Semantik jedes Topologieelements durch einen eindeutigen Typ spezifiziert.

Definition 19 (Komponententypen). Sei $d \in D$ ein DMMN-Modell, dann enthält die endliche Menge $KT_d := \pi_3(d) = \{kt_1, kt_2, \dots, kt_n \mid n \in \mathbb{N}\}$ alle Instanzen kt_i der DMMN-Klasse „Komponententyp“, welche jeweils einen Typ von Komponenten der Anwendung repräsentiert. Diese Instanzen werden als *Komponententypen des Modells* bezeichnet. Jedes $kt_i \in KT_d$ ist ein Paar:

$$KT_d \subseteq \Sigma^+ \times \mathbb{B}$$
$$kt_i = (Name, Abstrakt)$$

Dabei ist:

- $\pi_1(kt_i) = Name$, $Name \in \Sigma^+$, ein in KT_d eindeutiges, nicht-leeres Wort, das die Semantik des repräsentierten Komponententyps spezifiziert,
- $\pi_2(kt_i) = Abstrakt$, $Abstrakt \in \mathbb{B}$, ein Wahrheitswert, der angibt, ob der repräsentierte Typ eine real betreibbare Komponente beschreibt oder ob es sich um einen abstrakten, nicht instanziierten Typ handelt. ■

Äquivalent dazu wird die DMMN-Klasse „Relationentyp“ wie folgt definiert:

Definition 20 (Relationentypen). Sei $d \in D$ ein DMMN-Modell, dann enthält die endliche Menge $RT_d := \pi_4(d) = \{rt_1, rt_2, \dots, rt_n \mid n \in \mathbb{N}\}$ alle Instanzen rt_i der DMMN-Klasse „Relationentyp“, welche jeweils einen Typ von Relationen der Anwendung repräsentiert. Diese Instanzen werden als *Relationentypen des Modells* bezeichnet. Jedes $rt_i \in RT_d$ ist dabei ein Paar:

$$RT_d \subseteq \Sigma^+ \times \mathbb{B}$$
$$rt_i = (Name, Abstrakt)$$

Die Semantik der Attribute entspricht der der Komponententypen.

■

Die Komponenten- und Relationentypen eines DMMN-Modells beschreiben somit die Typen der repräsentierten Komponenten und Relationen. Zur übersichtlicheren Verwendung wird deren Vereinigung wie folgt definiert:

Definition 21 (Typelemente). Seien KT_d die Komponententypen und RT_d die Relationentypen eines DMMN-Modells $d \in D$, dann wird deren Vereinigung $TET_d := KT_d \cup RT_d$ als *Typelemente des Modells* bezeichnet. ■

Durch folgende Abbildung werden Topologieelemente, d. h. Komponenten- und Relationenelemente, auf deren jeweiliges Typelement abgebildet. Die Abbildung spezifiziert dadurch die Färbung der Knoten bzw. die Gewichtung der Kanten.

Definition 22 (Abbildung: typ_d). Seien TE_d Topologieelemente und TET_d Typelemente eines DMMN-Modells $d \in D$, dann bildet die Abbildung $typ_d := \pi_{12}(d)$ jedes Topologieelement $te_i \in TE_d$ auf genau ein Typelement $tet_j \in TET_d$ ab, welches die Semantik des Topologieelements beschreibt, oder auf $\perp$, falls für die repräsentierte Komponente bzw. Relation kein Typ spezifiziert / bekannt ist.

$$typ_d: TE_d \rightarrow TET_d \cup \{\perp\}$$

■

Typelemente können voneinander erben, um so verschiedene Abstraktionsschichten zu repräsentieren. Beispielsweise kann der konkrete Komponententyp *ApacheWebServer* die Semantik dessen abstrakten Supertyps *WebServer* verfeinern. Daher wird folgende Abbildung definiert, um Vererbungshierarchien von Komponententypen und Relationentypen abzubilden:

Definition 23 (Abbildung: $supertyp_d$). Sei TET_d die Menge aller Typelemente eines DMMN-Modells $d \in D$, dann bildet die Abbildung $supertyp_d := \pi_{13}(d)$ jedes Typelement $tet_i \in TET_d$ auf ein oder kein Typelement $tet_j \in TET_d$ ab, von dem es die Semantik erbt. Das Typelement tet_j heißt *Supertypelement* und repräsentiert den unmittelbaren Supertypen des von tet_i repräsentierten Typs.

$$supertyp_d: TET_d \rightarrow TET_d \cup \{\perp\}$$

Dabei gilt folgende Einschränkung:

$$\forall tet_i \in TET_d : tet_i \neq supertyp_d(tet_i)$$

Hat ein Typ keinen Supertyp, wird dies durch $\perp$ ausgedrückt. ■

Um die Menge aller Supertypen eines Topologieelements abzubilden, von denen es direkt oder transitiv erbt, wird folgende Abbildung definiert:

Definition 24 (Abbildung: supertypen). Seien TE_d Topologieelemente und TET_d Typelemente eines DMMN-Modells $d \in D$, dann bildet die Abbildung *supertypen* jedes Topologieelement $te_i \in TE_d$ auf alle Typelemente $tet_j \in TET_d$ ab, von denen es direkt oder transitiv erbt:

$$supertypen : TE_d \rightarrow \wp(TET_d), te_i \mapsto X_n$$

Dabei wird X_n wie folgt definiert:

$$\begin{aligned} X_0 &:= \{typ_d(te_i) \mid typ_d(te_i) \neq \perp\}, \\ X_k &:= X_{k-1} \cup \bigcup_{x \in X_{k-1}} \{supertyp_d(x) \mid supertyp_d(x) \neq \perp\}, \text{ für } k \geq 1 \end{aligned}$$

Die Sequenz $(X_k)_{k \in \mathbb{N}_0}$ ist monoton steigend. Da $X_k \subseteq TET_d$ und $|TET_d| < \infty$ wird $(X_k)_{k \in \mathbb{N}_0}$ stationär, d. h.

$$\exists n \in \mathbb{N}_0 : X_0 \subset X_1 \subset \dots \subset X_n = X_{n+1} = \dots$$

Bei der Modellierung von DMMN-Modellen und Typhierarchien gilt die Einschränkung, dass kein Typ direkt oder transitiv von sich selbst erben darf:

$$\forall tet_i \in TET_d : tet_i \notin supertypen(tet_i)$$

■

4.2.3 Technische Managementschicht

Die technische Managementschicht definiert Klassen zur Modellierung aller technischen Managementinformationen über Komponenten, Relationen und deren Typen. Alle Klassen dieser technischen Schicht haben Abhängigkeiten zu den Klassen „Topologieelement“ und „Topologieelementtyp“. Zur übersichtlicheren Formalisierung wird daher deren Vereinigungsmenge definiert:

Definition 25 (Managementspezifikationenelemente). Sei TE_d die endliche Menge aller Topologieelemente und TET_d die endliche Menge aller Typelemente eines DMMN-Modells $d \in D$, dann wird deren Vereinigung $MS_d := TE_d \cup TET_d$ als *Managementspezifikationenelemente des Modells* bezeichnet. ■

4.2.3.1 Eigenschaften

In DMMN-Modellen kann ein Topologieelement sowohl (i) *deskriptiv* den gegenwärtigen Zustand bzw. die gegenwärtige Konfiguration einer existierenden Komponente oder Relation repräsentieren sowie (ii) *präskriptiv* den gewünschten Zielzustand bzw. die gewünschte Zielkonfiguration einer existierenden oder nicht existenten, d. h. bereitzustellenden, Komponente oder Relation spezifizieren. Die Modellierung dieser Informationen ermöglichen Instanzen der DMMN-Klasse „Eigenschaft“. Das DMMN-Metamodell kombiniert dadurch die beiden Metamodelltypen des Instanzmodells und des Zielzustandsmodells. Zudem können Instanzen dieser Klasse verwendet werden, um Standardwerte (engl. „*default values*“) eines Komponenten- bzw. Relationentyps zu beschreiben.

Definition 26 (Eigenschaftenelemente). Sei $d \in D$ ein DMMN-Modell, dann enthält die endliche Menge $E_d := \pi_5(d) = \{e_1, e_2, \dots, e_n \mid n \in \mathbb{N}\}$ alle Instanzen e_i der DMMN-Klasse „Eigenschaft“, welche jeweils entweder (i) eine gegenwärtige Eigenschaft einer existierenden Komponente oder Relation repräsentiert, (ii) eine gewünschte Eigenschaft einer Komponente oder Relation beschreibt oder (iii) eine Standardeigenschaft eines Komponenten- oder Relationentyps spezifiziert. Diese Instanzen werden als *Eigenschaftenelemente des Modells* bezeichnet.

Jedes Eigenschaftenelement $e_i \in E_d$ ist ein Quadrupel:

$$E_d \subseteq \Sigma^+ \times \Sigma^+ \times (\Sigma^* \cup \{\perp, *\}) \times \Sigma^+ \\ e_i = (Id, Name, Wert, Datentyp)$$

Dabei ist:

- $\pi_1(e_i) = Id$, $Id \in \Sigma^+$, ein innerhalb E_d eindeutiges, nicht-leeres Wort, das die dargestellte bzw. gewünschte Eigenschaft identifiziert,
- $\pi_2(e_i) = Name$, $Name \in \Sigma^+$, ein nicht-leeres Wort, das den Namen der dargestellten bzw. gewünschten Eigenschaft repräsentiert,
- $\pi_3(e_i) = Wert$, $Wert \in \Sigma^* \cup \{\perp, *\}$, ein Wort, das den Wert der dargestellten bzw. gewünschten Eigenschaft repräsentiert. Falls $Wert = \perp$, dann ist der repräsentierte Wert undefiniert. Falls $Wert = *$, dann existiert ein Wert, der unbekannt ist oder nicht im Modell repräsentiert werden soll,
- $\pi_4(e_i) = Datentyp$, $Datentyp \in \Sigma^+$, ein nicht-leeres Wort, das den Datentyp des Werts der dargestellten bzw. gewünschten Eigenschaft repräsentiert.

Es gilt die Eindeutigkeit des Identifikators innerhalb des DMMN-Modells d :

$$\forall e_1, e_2 \in E_d: \pi_1(e_1) = \pi_1(e_2) \Rightarrow e_1 = e_2 \quad \blacksquare$$

Die folgende Abbildung weist Komponenten-, Relationen- und Typelementen deren Eigenschaftenelemente zu:

Definition 27 (Abbildung: *eigeneEigenschaften_d*). Sei MS_d die Menge aller Managementspezifikationenelemente und E_d die Menge aller Eigenschaftenelemente eines DMMN-Modells $d \in D$, dann bildet die folgende Abbildung $eigeneEigenschaften_d := \pi_{14}(d)$ jedes Element $ms_i \in MS_d$ auf dessen eigene Eigenschaftenelemente ab, welche die Eigenschaften der durch ms_i repräsentierten Komponente, Relation oder des durch ms_i repräsentierten Typs beschreiben.

$$eigeneEigenschaften_d: MS_d \rightarrow \wp(E_d)$$

Dabei darf jeder Eigenschaftsname innerhalb eines Managementspezifikationsenelements $ms_j \in MS_d$ höchstens einmal verwendet werden:

$$\forall ms_j \in MS_d \forall e_1, e_2 \in \text{eigeneEigenschaften}_d(ms_j): \pi_2(e_1) = \pi_2(e_2) \Rightarrow e_1 = e_2$$

■

Die Menge aller Eigenschaftenelemente aller DMMN-Modelle wird definiert als:

Definition 28 (Menge aller Eigenschaftenelemente). Sei $D = \{d_1, \dots, d_n \mid n \in \mathbb{N}\}$ die Menge aller DMMN-Modelle, E_{d_i} die Menge aller Eigenschaftenelemente eines DMMN-Modells $d_i \in D$, dann enthält die endliche Menge $E := \{e_1, e_2, \dots, e_n \mid n \in \mathbb{N}\}$ alle Eigenschaftenelemente aller DMMN-Modelle $d_i \in D$. Diese wird als *Menge aller Eigenschaftenelemente* bezeichnet:

$$E := \bigcup_{d_i \in D} E_{d_i}$$

■

Neben den eigenen Eigenschaften einer Komponente oder Relation erben diese alle Eigenschaften ihres Typs und all ihrer Supertypen. Spezifizieren Komponente bzw. Relation und deren Typ Eigenschaften mit identischem Namen, dann gilt die Eigenschaft der Komponente bzw. Relation. Gleichnamige Eigenschaften eines Typs überschreiben die ihres Supertyps. Um alle Eigenschaften eines Topologieelements, welche sich durch Vererbung gemäß der beschriebenen Vererbungsgesetze ergeben, abzubilden, wird die Hilfsfunktion *eigenschaften* eingeführt. Dafür wird zunächst der generische Vererbungsoperator $\uplus$ definiert, welcher die beschriebenen Vererbungsgesetze generisch formalisiert.

Definition 29 (Vererbungsoperator $\uplus_i$). Seien $X_1, \dots, X_n$ nicht-leere Mengen, $X \subseteq X_1 \times \dots \times X_n, n \geq i, n \in \mathbb{N}, i \in \mathbb{N}$, und $A, B \subseteq X$ beliebige Teilmengen von X , dann wird der Vererbungsoperator $\uplus_i$ wie folgt definiert:

$$A \uplus_i B := \left\{ x \mid (x \in A) \vee (x \in B \wedge \nexists y \in A : \pi_i(x) = \pi_i(y)) \right\}$$

Für $i = 2$ wird der Vererbungsoperator ohne Subskript notiert, d. h. $\uplus$.

■

Der Vererbungsoperator $\uplus$ bildet die oben beschriebenen Vererbungsgesetze somit unter der Annahme ab, dass das vererbungsrelevante Attribut jedes Elements $x \in X$ durch die zweite Komponente des Tupels spezifiziert wird. Sei z. B. $X = E$ die Menge aller Eigenschaftenelemente, $A \subset X$ die Menge der Eigenschaftenelemente eines Komponentenelements des Typs *ApacheWebServer* und $B \subset X$ die Menge der Eigenschaftenelemente dieses Typs. Enthält B ein Eigenschaftenelement namens *HTTP-Port*, spezifiziert dieses einen Standardwert, der im Falle eines gleichnamigen Elements in A von diesem überschrieben wird. Enthält A kein gleichnamiges Element, überträgt sich die Eigenschaft des Typs auf das Komponentenelement. Im Folgenden wird das Prinzip der mehrfachen Verkettung einer Abbildung verwendet, welches wie folgt definiert ist:

Definition 30 (Verkettung). Sei $f : X \rightarrow X$ eine Abbildung, deren Definitionsbereich dem Bildbereich entspricht, d. h. eine homogene Relation. Die wiederholte Komposition einer homogenen Relation wird als *Verkettung* bezeichnet. Für eine k-fache Wiederholung der Komposition wird folgende Schreibweise definiert:

$$\underbrace{f \circ \dots \circ f}_k = f^k$$

■

Definition 31 (Abbildung: eigenschaften). Sei TE_d die Menge aller Topologieelemente und E_d die Menge aller Eigenschaftenelemente eines DMMN-Modells $d \in D$, dann bildet die Abbildung *eigenschaften* jedes Element $te_i \in TE_d$ unter Einhaltung der Vererbungsgesetze auf dessen Eigenschaftenelemente ab:

$$eigenschaften : TE_d \rightarrow \wp(E_d), te_i \mapsto X_n$$

Dabei wird X_n wie folgt definiert:

$$n := |supertypen(te_i)|$$

$$X_0 := eigeneEigenschaften_d(te_i)$$

$$X_k := X_{k-1} \uplus eigeneEigenschaften_d(supertyp_d^{k-1}(typ_d(te_i))), \text{ für } n \geq k \geq 1$$

■

4.2.3.2 Managementschnittstellen und Managementoperationen

Zum Management von Komponenten und Relationen werden verschiedenste Managementtechnologien eingesetzt, beispielsweise ausführbare Skripte oder deklarative Werkzeuge zum Konfigurationsmanagement von Software (vgl. [Abschnitt 2.3](#)). Um die Managementoperationen, welche von diesen Technologien für eine bestimmte Komponente oder Relation der Anwendung angeboten werden, formal funktional zu beschreiben, definiert das DMMN-Metamodell die Klassen „Managementschnittstelle“ und „Managementoperation“.

Definition 32 (Managementschnittstellenelemente). Sei $d \in D$ ein DMMN-Modell, dann enthält die endliche Menge $I_d := \pi_6(d) = \{i_1, i_2, \dots, i_n \mid n \in \mathbb{N}\}$ alle Instanzen i_i der DMMN-Klasse „Managementschnittstelle“, welche jeweils eine Zusammenfassung verfügbarer Managementoperationen zum Management der Anwendung repräsentiert und deren Semantik definiert. Diese Instanzen werden als *Managementschnittstellenelemente des Modells* bezeichnet. Jedes Managementschnittstellenelement $i_i \in I_d$ ist dabei als Paar definiert:

$$I_d \subseteq \Sigma^+ \times \Sigma^+ \\ i_i = (Id, Typ)$$

Dabei ist:

- $\pi_1(i_i) = Id, Id \in \Sigma^+$, ein innerhalb I_d eindeutiges, nicht-leeres Wort, das die repräsentierte Schnittstelle eindeutig identifiziert,
- $\pi_2(i_i) = Typ, Typ \in \Sigma^+$, ein nicht-leeres Wort, das den Typ, d. h. die Semantik, der repräsentierten Managementschnittstelle beschreibt.

■

Managementschnittstellen fassen Operationen zusammen, die für eine Komponente bzw. Relation ausgeführt und aufgrund ihrer Funktionalität logisch gruppiert werden können. Um die individuellen Schnittstellen einer bestimmten Komponente oder Relation bzw. die Schnittstellen, welche auf Basis deren Typen angeboten werden, zu modellieren, wird folgende Abbildung definiert:

Definition 33 (Abbildung: *eigeneSchnittstellen_d*). Sei $MS_d := TE_d \cup TET_d$ die Menge aller Managementspezifikationenelemente und I_d die Menge aller Managementschnittstellenelemente eines DMMN-Modells $d \in D$, dann bildet die Abbildung $eigeneSchnittstellen_d := \pi_{15}(d)$ jedes Topologie- bzw. Typelement $ms_i \in MS_d$ auf dessen eigene Schnittstellenelemente ab, welche die individuellen Managementschnittstellen der durch ms_i repräsentierten Komponente oder Relation beschreiben bzw. die Schnittstellen des von ms_i repräsentierten Typs spezifizieren, die generell für diesen Typ verfügbar sind.

$$eigeneSchnittstellen_d: MS_d \rightarrow \wp(I_d)$$

Dabei darf jeder Managementschnittstellentyp innerhalb eines Managementspezifikationenelements $ms_j \in MS_d$ höchstens einmal verwendet werden:

$$\begin{aligned} \forall ms_j \in MS_d \forall i_1, i_2 \in eigeneSchnittstellen_d(ms_j): \\ (\pi_1(i_1) = \pi_1(i_2) \Rightarrow i_1 = i_2) \wedge \\ (\pi_2(i_1) = \pi_2(i_2) \Rightarrow i_1 = i_2) \end{aligned}$$

■

Die Menge aller Schnittstellenelemente aller DMMN-Modelle wird definiert als:

Definition 34 (Menge aller Managementschnittstellenelemente). Sei $D = \{d_1, \dots, d_n \mid n \in \mathbb{N}\}$ die Menge aller DMMN-Modelle, I_{d_i} die Menge aller Schnittstellenelemente eines DMMN-Modells $d_i \in D$, dann enthält die endliche Menge $I := \{i_1, i_2, \dots, i_n \mid n \in \mathbb{N}\}$ alle Schnittstellenelemente aller DMMN-Modelle $d_i \in D$. Diese wird als *Menge aller Schnittstellenelemente* bezeichnet:

$$I := \bigcup_{d_i \in D} I_{d_i}$$

■

Eine Managementschnittstelle definiert dabei mindestens eine Managementoperation, die zum Management einer Komponente oder Relation der Anwendung ausgeführt werden kann. Managementschnittstellen und deren Operationen können dabei generell durch beliebige Technologien implementiert sein. Während Cloud-Anbieter Operationen zum Management der angebotenen Dienste typischerweise in Form aufrufbarer APIs anbieten, können Managementoperationen auch durch eigenständige Skripte oder Programme implementiert werden, beispielsweise, um einen Webserver automatisiert durch Ausführung eines Skripts auf einem Betriebssystem zu installieren. Zur Ausführung von Operationen können Instanzinformationen der Anwendung erforderlich sein, welche als Eingabe übergeben werden. Zum Beispiel benötigt eine Operation, welche eine Verbindung zu einer Datenbank herstellt, deren Zugangsdaten und Endpunkt. Andererseits beeinflusst die Ausführung einer Operation häufig den Zustand von Komponenten und Relationen, wodurch Ausgaben einer Operation häufig Instanzinformationen darstellen. Um die Signatur sowie die operative Semantik einer Managementoperation innerhalb einer Managementschnittstelle unabhängig von deren technischer Implementierung zu beschreiben, wird die DMMN-Klasse „Managementoperation“ eingeführt:

Definition 35 (Managementoperationenelemente). Sei $d \in D$ ein DMMN-Modell, dann enthält die endliche Menge $O_d := \pi_7(d) = \{o_1, o_2, \dots, o_n \mid n \in \mathbb{N}\}$ alle Instanzen o_i der DMMN-Klasse „Managementoperation“ des Modells, welche jeweils die Signatur einer ausführbaren Managementoperation einer im Modell repräsentierten Managementschnittstelle beschreibt. Diese Instanzen werden als *Managementoperationenelemente des Modells* bezeichnet. Jedes Managementoperationenelement $o_i \in O_d$ ist als Quadrupel definiert:

$$O_d \subseteq \Sigma^+ \times \Sigma^+ \times \wp(\text{Parameter}) \times \wp(\text{Parameter})$$

$$o_i = (\text{Id}, \text{Typ}, \text{Eingabeparameter}, \text{Ausgabeparameter})$$

Dabei ist:

- $\pi_1(o_i) = \text{Id}, \text{Id} \in \Sigma^+$, ein innerhalb O_d eindeutiges, nicht-leeres Wort, das die repräsentierte Operation identifiziert,

- $\pi_2(o_i) = \text{Typ}, \text{Typ} \in \Sigma^+$, ein nicht-leeres Wort, das den Typ der repräsentierten Operation beschreibt und dessen Semantik definiert,
- $\pi_3(o_i) = \text{Eingabeparameter}, \text{Eingabeparameter} \subseteq \text{Parameter}$, eine Menge, deren Elemente die Eingabeparameter der Operation repräsentieren,
- $\pi_4(o_i) = \text{Ausgabeparameter}, \text{Ausgabeparameter} \subseteq \text{Parameter}$, eine Menge, deren Elemente die Ausgabeparameter der Operation repräsentieren.

Die Elemente der Menge *Parameter* repräsentieren folglich sowohl die Eingabe- als auch die Ausgabeparameter von Managementoperationen. Diese werden durch einen eindeutigen Namen identifiziert, nicht durch deren Reihenfolge. Jedes *Parameterelement* $\text{param}_j \in \text{Parameter}$ ist dabei als Tripel definiert:

$$\begin{aligned} \text{Parameter} &\subseteq \Sigma^+ \times ((TE_d \times E_d) \cup \{\perp\}) \times \Sigma^+ \\ \text{param}_j &:= (\text{Name}, \text{Eigenschaftsreferenz}, \text{Datentyp}) \end{aligned}$$

Dabei ist:

- $\pi_1(\text{param}_j) = \text{Name}, \text{Name} \in \Sigma^+$, ein nicht-leeres Wort, welches den Namen des Parameters repräsentiert
- $\pi_2(\text{param}_j) = \text{Eigenschaftsreferenz}, \text{Eigenschaftsreferenz} \in (TE_d \times E_d) \cup \{\perp\}$, die Repräsentation einer Referenz, welche auf eine Eigenschaft einer Komponente bzw. Relation zeigt. Sei $\pi_2(\text{param}_j) = (te_i, e_p)$, dann repräsentiert te_i die Komponente bzw. Relation und e_p deren referenzierte Eigenschaft. Bei der Ausführung der von o_i repräsentierten Operation gilt:

Falls $\text{param}_j \in \pi_3(o_i) \wedge \pi_2(\text{param}_j) = (te_i, e_p)$, dann muss der gegenwärtige Wert der durch (te_i, e_p) referenzierten Eigenschaft bei Ausführung der von o_i repräsentierten Operation als Eingabewert für den von param_j repräsentierten Eingabeparameter verwendet werden.

Falls $\text{param}_j \in \pi_4(o_i) \wedge \pi_2(\text{param}_j) = (te_i, e_p)$, dann muss der zurückgegebene Wert des von param_j repräsentierten Ausgabeparameters nach Ausführung der Operation, welche von o_i repräsentiert wird, als Wert für die durch (te_i, e_p) repräsentierte Eigenschaft gesetzt werden.

Ist $\pi_2(param_j) = \perp$, dann müssen keine Eigenschaften bezüglich des von $param_j$ repräsentierten Parameters beachtet werden und der Aufrufende kann den Parameterwert frei wählen. Dies stellt den Regelfall dar.

- $\pi_3(param_j) = Datentyp, Datentyp \in \Sigma^+$, ein nicht-leeres Wort, das den Datentyp des repräsentierten Parameters beschreibt.

Es gilt die Eindeutigkeit der Parameternamen:

$$\forall param_1, param_2 \in \pi_3(o_i): \pi_1(param_1) = \pi_1(param_2) \Rightarrow param_1 = param_2$$

$$\forall param_1, param_2 \in \pi_4(o_i): \pi_1(param_1) = \pi_1(param_2) \Rightarrow param_1 = param_2$$



Innerhalb einer Managementschnittstelle ermöglicht das Attribut „Typ“ eines Operationenelements die eindeutige Spezifikation der Semantik der repräsentierten Operation. Beispielsweise kann die Semantik einer Operation des Typs *Install* innerhalb einer Managementschnittstelle des Typs *LifecycleManagement* eindeutig bestimmt werden, sofern die Semantik für beide Typen definiert ist.

Die Möglichkeit, Eigenschaftenreferenzen für Parameter zu spezifizieren, ermöglicht die Modellierung des Datenflusses auf Basis der Anwendungstopologie. Durch dieses Konzept können Informationen, welche erst zur Laufzeit bekannt sind, direkt im DMMN-Modell als Eingaben für Operationen spezifiziert werden bzw. Resultate den Instanzdaten von Komponenten und Relationen zugewiesen werden. Gemäß des anfangs erwähnten Beispiels der Verbindung einer Anwendung mit einer Datenbank, die z. B. in unterschiedlichen Cloud-Umgebungen betrieben werden, wird eine Operation benötigt, welche den Endpunkt der Datenbank in die Konfiguration der Anwendung überträgt. Da dieser Endpunkt erst während der Bereitstellung bekannt wird, kann eine Eigenschaftsreferenz verwendet werden, um die benötigte Information zu referenzieren. Managementoperationen, welche zur Kompensation bereits abgeschlossener Managementdurchführungen dienen, dürfen keinerlei Eigenschaftsreferenzen für Parameter deklarieren. Um einer Managementschnittstelle ihre Managementoperationen zuzuordnen, wird folgende Abbildung definiert:

Definition 36 (Abbildung: $operationen_d$). Sei I_d die Menge aller Managementschnittstellenelemente und O_d die Menge aller Operationenelemente eines DMMN-Modells $d \in D$, dann bildet die Abbildung $operationen_d := \pi_{16}(d)$ jedes Managementschnittstellenelement $i_i \in I_d$ auf dessen Managementoperationenelemente ab, welche die Operationen der durch i_i beschriebenen Managementschnittstelle repräsentieren.

$$operationen_d: I_d \rightarrow \wp(O_d)$$

Dabei darf jeder Operationentyp innerhalb eines Managementschnittstellenelements $i_j \in I_d$ höchstens einmal verwendet werden:

$$\forall i_j \in I_d \forall o_1, o_2 \in operationen_d(i_j): (\pi_2(o_1) = \pi_2(o_2) \Rightarrow o_1 = o_2)$$

■

Die Menge aller Operationenelemente aller DMMN-Modelle wird definiert als:

Definition 37 (Menge aller Managementoperationenelemente). Sei $D = \{d_1, \dots, d_n \mid n \in \mathbb{N}\}$ die Menge aller DMMN-Modelle, O_{d_i} die Menge aller Managementoperationenelemente eines DMMN-Modells $d_i \in D$, dann enthält die endliche Menge $O := \{o_1, o_2, \dots, o_n \mid n \in \mathbb{N}\}$ alle Managementoperationenelemente aller DMMN-Modelle $d_i \in D$. Diese wird als *Menge aller Managementoperationenelemente* bezeichnet:

$$O := \bigcup_{d_i \in D} O_{d_i}$$

■

Managementschnittstellen werden bei der Typisierung von Komponenten und Relationen vererbt. Haben Managementschnittstellen identische Typen, überschreibt die Schnittstelle der Komponente bzw. Relation die des Typs, sowie die Schnittstelle eines Subtyps die des Supertyps. Um alle Schnittstellenelemente, welche auf Basis dieser Vererbungsgesetze von einem Topologieelement angeboten werden, abzubilden, wird folgende Abbildung definiert:

Definition 38 (Abbildung: *schnittstellen*). Sei TE_d die Menge aller Topologieelemente und I_d die Menge aller Schnittstellenelemente eines DMMN-Modells $d \in D$, dann bildet die Abbildung *schnittstellen* jedes Element $te_i \in TE_d$ unter Einhaltung der Vererbungsgesetze auf dessen Schnittstellenelemente ab:

$$schnittstellen: TE_d \rightarrow \wp(I_d), te_i \mapsto X_n$$

Dabei wird X_n wie folgt definiert:

$$n := |supertypen(te_i)|$$

$$X_0 := eigeneSchnittstellen_d(te_i)$$

$$X_k := X_{k-1} \uplus eigeneSchnittstellen_d(supertyp_d^{k-1}(typ_d(te_i))), \text{ für } n \geq k \geq 1$$

■

Um Managementoperationen auszuführen, müssen diesen Implementierungen zugeordnet werden. Dazu werden im nächsten Abschnitt Artefakte eingeführt.

4.2.3.3 Artefaktelemente

Bei der Bereitstellung und dem Management von Anwendungen sind generell zwei unterschiedliche Typen an Artefakten involviert: (i) *Anwendungsartefakte* und (ii) *Operationsimplementierungen*. Anwendungsartefakte implementieren Komponenten und Relationen einer Anwendung und werden zu deren Ausführung benötigt. Beispiele sind PHP-Dateien, die von einem Webserver betrieben werden, um die Anwendungsfunktionalität bereitzustellen oder auch SQL-Dateien, welche bei der Anwendungsbereitstellung als initiale Datenbasis in eine Datenbank eingespielt werden. In ihrer Summe implementieren Anwendungsartefakte somit die gesamte Funktionalität einer Anwendung.

Eine Operationsimplementierung beschreibt die ausführbare Realisierung einer bestimmten Managementoperation einer Managementschnittstelle und wird somit zum Management einer Komponente bzw. Relation aufgerufen. Eine

Operationsimplementierung kann beispielsweise durch ein Skript realisiert sein, dessen Ausführung den Aufruf der assoziierten Managementoperation implementiert. Operationsimplementierungen können jedoch auch nur Informationen über eine verfügbare Managementoperation einer Schnittstelle beschreiben, beispielsweise den Endpunkt eines aufrufbaren Managementdienstes eines Web-basierten API. Durch dieses Konzept können sowohl Komponenten als auch Relationen Managementschnittstellen definieren, welche die eigentlichen Entitäten nicht nativ bereitstellen, jedoch anderweitig zur Verfügung gestellt werden. Um beide Arten von Artefakten formal zu beschreiben, definiert das DMMN-Metamodell die Klassen „Anwendungsartefakt“ und „Operationsimplementierung“. Da beide Artefakttypen identisch formalisiert werden und sich nur in ihrer Semantik unterscheiden, wird deren Superklasse „Artefakt“ als Basis der Definition verwendet und die jeweiligen Instanzen der verschiedenen Klassen anschließend durch eine Abbildung ihrer Semantik zugeordnet.

Definition 39 (Artefaktelemente). Sei $d \in D$ ein DMMN-Modell, dann enthält die endliche Menge $A_d := \pi_8(d) = \{a_1, a_2, \dots, a_n \mid n \in \mathbb{N}\}$ alle Instanzen a_i der DMMN-Klasse „Artefakt“, welche jeweils entweder ein Anwendungsartefakt der Anwendung oder eine Operationsimplementierung repräsentiert. Diese Instanzen werden als *Artefaktelemente des Modells* bezeichnet. Jedes Artefaktelement $a_i \in A_d$ ist dabei ein Quadrupel:

$$A_d \subseteq \Sigma^+ \times (\Sigma^+ \cup \{\perp\}) \times \Sigma^* \times (\Sigma^+ \cup \{\perp\})$$

$$a_i = (Id, Name, Daten, Datentyp)$$

Dabei ist:

- $\pi_1(a_i) = Id, Id \in \Sigma^+$, ein innerhalb A_d eindeutiges, nicht-leeres Wort, welches das repräsentierte Artefakt eindeutig identifiziert,
- $\pi_2(a_i) = Name, Name \in \Sigma^+ \cup \{\perp\}$, ein Wort, das den Namen des dargestellten Artefakts repräsentiert. Falls $Name = \perp$, dann ist der Name des repräsentierten Artefakts undefiniert,
- $\pi_3(a_i) = Daten, Daten \in \Sigma^*$, ein möglicherweise leeres Wort, das die Daten des dargestellten Artefakts repräsentiert,

- $\pi_4(a_i) = \text{Datentyp}$, $\text{Datentyp} \in \Sigma^+ \cup \{\perp\}$, ein Wort, das den Datentyp der Daten des dargestellten Artefakts repräsentiert. Falls $\text{Datentyp} = \perp$, dann ist der Typ der Daten des repräsentierten Artefakts undefiniert. ■

Die Implementierung eines Artefakts bzw. Informationen über ein Artefakt können in Form beliebiger „Daten“ beschrieben werden. Wird beispielsweise eine Skriptimplementierung spezifiziert, kann das „Typ“-Attribut diese Daten typisieren. Um Komponenten, Relationen und Typen die zugehörigen Anwendungsartefakte sowie Managementoperationen deren Operationsimplementierungen zuzuordnen, wird folgende Abbildung definiert:

Definition 40 (Abbildung: eigeneArtefakte_d). Sei MS_d die Menge aller Managementspezifikationenelemente, O_d die Menge aller Operationenelemente und A_d die Menge aller Artefaktelemente eines DMMN-Modells $d \in D$, dann weist die Abbildung $\text{eigeneArtefakte}_d := \pi_{17}(d)$ jedem Managementspezifikationenelement $ms_i \in MS_d$ die Artefaktelemente zu, welche die Anwendungsartefakte der von ms_i repräsentierten Komponente, Relation oder des repräsentierten Typs beschreiben, und jedem Operationenelement $o_j \in O_d$ die Artefaktelemente zu, welche Operationsimplementierungen der von o_j repräsentierten Managementoperation beschreiben.

$$\text{eigeneArtefakte}_d : MS_d \cup O_d \rightarrow \wp(A_d) \quad \blacksquare$$

Die Menge aller Artefaktelemente aller DMMN-Modelle wird definiert als:

Definition 41 (Menge aller Artefaktelemente). Sei $D = \{d_1, \dots, d_n \mid n \in \mathbb{N}\}$ die Menge aller DMMN-Modelle, A_{d_i} die Menge aller Artefaktelemente eines DMMN-Modells $d_i \in D$, dann enthält die endliche Menge $A := \{a_1, a_2, \dots, a_n \mid n \in \mathbb{N}\}$ alle Artefaktelemente aller DMMN-Modelle $d_i \in D$. Diese wird als *Menge aller Artefaktelemente* bezeichnet:

$$A := \bigcup_{d_i \in D} A_{d_i} \quad \blacksquare$$

Dabei gelten für Anwendungsartefakte dieselben Vererbungsgesetze wie für Eigenschaften und Schnittstellen: Bei der Vererbung von Topologieelementen gilt für Anwendungsartefakte mit gleichem Namen, welcher für jedes Artefakt $a_i \in A_d$ durch die Projektionsfunktion $\pi_2(a_i)$ abgebildet wird, dass die Artefakte des Subtyps die des Supertyps überschreiben. Dadurch können Vererbungshierarchien genutzt werden, um individuelle Implementierungen von Komponenten bzw. Relationen auf Basis bekannter Typen zu verfeinern.

Definition 42 (Abbildung: artefakte). Sei TE_d die Menge aller Topologieelemente und A_d die Menge aller Artefaktelemente eines DMMN-Modells $d \in D$, dann bildet die Abbildung *artefakte* jedes Element $te_i \in TE_d$ unter Einhaltung der Vererbungsgesetze auf dessen Artefaktelemente ab, welche Anwendungsartefakte der in d beschriebenen Anwendung repräsentieren:

$$\text{artefakte}: TE_d \rightarrow \wp(A_d), te_i \mapsto X_n$$

Dabei wird X_n wie folgt definiert:

$$n := | \text{supertypen}(te_i) |$$

$$X_0 := \text{eigeneArtefakte}_d(te_i)$$

$$X_k := X_{k-1} \uplus \text{eigeneArtefakte}_d(\text{supertyp}_d^{k-1}(\text{typ}_d(te_i))), \text{ für } n \geq k \geq 1$$

■

Da Operationsimplementierungen direkt einer Operation zugeordnet sind, welche nicht explizit sondern nur implizit über Managementschnittstellen vererbt werden, gelten für Operationsimplementierungen keine Vererbungsgesetze: Überschreibt eine Managementschnittstelle die des Typs bzw. des Supertyps, werden die Operationen der überschreibenden Managementschnittstelle inklusive der jeweils zugehörigen Operationsimplementierungen übernommen.

4.2.4 Deklarative Managementschicht

Deklaratives Anwendungsmanagement erfordert die deklarative Modellierung auszuführender Managementaufgaben (vgl. [Definition 6](#)). Dazu definiert die deklarative Managementschicht des DMMN-Metamodells die folgenden Klassen: „Managementannotation“, „Konfiguration“ und „Reihenfolgespezifikation“.

4.2.4.1 Managementannotationen und Konfigurationen

Das DMMN-Metamodell ermöglicht die deklarative Modellierung auszuführender Managementaufgaben in Abhängigkeit des Instanzmodells der Anwendung, welches durch Elemente der Topologie- und technischen Managementschicht beschrieben wird. Beispielsweise, um eine Anwendung auf einen Webserver auszuliefern, eine Datenbank auf einem Betriebssystem zu installieren oder Daten in eine Datenbank zu importieren. Zustandsmodifizierende sowie zustandserhaltende Managementaufgaben werden dabei deklarativ durch Instanzen der DMMN-Klasse „Managementannotation“ beschrieben ([Anforderung 1](#)).

Definition 43 (Managementannotationenelemente). Sei $d \in D$ ein DMMN-Modell, dann enthält die endliche Menge $MA_d := \pi_9(d) = \{ma_1, ma_2, \dots, ma_n \mid n \in \mathbb{N}\}$ alle Instanzen ma_i der DMMN-Klasse „Managementannotation“, welche jeweils eine auszuführende Managementaufgabe repräsentiert. Diese Instanzen werden als *Managementannotationenelemente des Modells* bezeichnet. Jedes Managementannotationenelement $ma_i \in MA_d$ ist dabei ein Paar:

$$\begin{aligned} MA_d &\subseteq \Sigma^+ \times \Sigma^+ \\ ma_i &= (Id, Typ) \end{aligned}$$

Dabei ist:

- $\pi_1(ma_i) = Id, Id \in \Sigma^+$, ein global eindeutiges, nicht-leeres Wort, das die repräsentierte Managementaufgabe identifiziert,
- $\pi_2(ma_i) = Typ, Typ \in \Sigma^+$, ein nicht-leeres Wort, das den Typ der repräsentierten Managementaufgabe und deren Semantik spezifiziert. ■

Aufgrund der deklarativen Natur von DMMN wird durch ein Managementannotationenelement nur beschrieben *was* umgesetzt werden soll, ohne dabei weitere Informationen anzugeben, *wie* die repräsentierte Managementaufgabe durchzuführen ist. Eine auszuführende Managementaufgabe wird dabei direkt mit der Komponente bzw. Relation assoziiert, für die sie ausgeführt werden soll. Dazu wird folgende Abbildung definiert:

Definition 44 (Abbildung: $annotationen_d$). Sei TE_d die Menge aller Topologieelemente und MA_d die Menge aller Annotationenelemente eines DMMN-Modells $d \in D$, dann bildet die Abbildung $annotationen_d := \pi_{18}(d)$ jedes Element $te_i \in TE_d$ auf alle Annotationenelemente ab, welche die Managementaufgaben repräsentieren, die bezüglich der von te_i repräsentierten Komponente bzw. Relation auszuführen sind.

$$annotationen_d: TE_d \rightarrow \wp(MA_d)$$

■

Die Menge aller Annotationenelemente aller DMMN-Modelle wird definiert als:

Definition 45 (Menge aller Annotationenelemente). Sei $D = \{d_1, \dots, d_n \mid n \in \mathbb{N}\}$ die Menge aller DMMN-Modelle, MA_{d_i} die Menge aller Annotationenelemente eines DMMN-Modells $d_i \in D$, dann enthält die endliche Menge $MA := \{ma_1, ma_2, \dots, ma_n \mid n \in \mathbb{N}\}$ alle Annotationenelemente aller DMMN-Modelle $d_i \in D$. Diese wird als *Menge aller Annotationenelemente* bezeichnet:

$$MA := \bigcup_{d_i \in D} MA_{d_i}$$

■

Zur exakten Beschreibung einer Managementaufgabe werden gegebenenfalls weitere Informationen benötigt. Soll beispielsweise die Managementaufgabe des Exports von Daten aus einer Datenbank beschrieben werden, reicht das Attribut „Typ“ allein nicht aus. Um die Aufgabe vollständig zu beschreiben, müssen weitere Informationen angegeben werden, welche beispielsweise die zu exportierenden Daten spezifizieren und das Ziel des Exports inklusive nötiger Zugangsdaten für das Speichermedium beschreiben. Diese Informationen werden als *Konfiguration* einer Managementaufgabe bezeichnet und können durch Instanzen der gleichnamigen Klasse „Konfiguration“ beschrieben werden:

Definition 46 (Konfigurationenelemente). Sei $d \in D$ ein DMMN-Modell, dann enthält die endliche Menge $C_d := \pi_{10}(d) = \{c_1, c_2, \dots, c_n \mid n \in \mathbb{N}\}$ alle Instanzen c_i der DMMN-Klasse „Konfiguration“, welche jeweils eine Konfiguration einer Managementaufgabe repräsentiert. Diese Instanzen werden als *Konfigurationenelemente des Modells* bezeichnet. Jedes Element $c_i \in C_d$ ist ein Tripel:

$$C_d \subseteq \Sigma^+ \times (\Sigma^+ \cup \{*\}) \times \wp(\Sigma^+ \times \Sigma^*)$$

$$c_i = (Id, Typ, Konfigurationsparameter)$$

Dabei ist:

- $\pi_1(c_i) = Id, Id \in \Sigma^+,$ ein innerhalb C_d eindeutiges, nicht-leeres Wort, welches die repräsentierte Konfiguration identifiziert,
- $\pi_2(c_i) = Typ, Typ \in \Sigma^+ \cup \{*\},$ ein nicht-leeres Wort, das den Typ der repräsentierten Konfiguration und deren Semantik definiert. Falls $Typ = *,$ dann ist der Typ der Konfiguration nicht eindeutig spezifiziert,
- $\pi_3(c_i) = Konfigurationsparameter, Konfigurationsparameter \in \wp(\Sigma^+ \times \Sigma^*),$ die Menge der Konfigurationsparameter, welche die repräsentierte Konfiguration parametrisieren. Sei $(n, w) \in \pi_3(c_i)$ ein repräsentierter Parameter, dann beschreibt n dessen Namen und w dessen Wert. ■

Konfigurationsparameter können auch zur Spezifikation von Zielzuständen verwendet werden, die durch Ausführung der zugehörigen Managementannotation für die jeweilige Komponente bzw. Relation erreicht werden sollen.

Um einer Managementaufgabe innerhalb eines DMMN-Modells $d \in D$ ihre Konfiguration zuzuweisen, wird die Abbildung $konfiguration_d$ definiert:

Definition 47 (Abbildung: $konfiguration_d$). Sei MA_d die Menge aller Annotationenelemente und C_d die Menge aller Konfigurationenelemente eines DMMN-Modells $d \in D$, dann bildet die Abbildung $konfiguration_d := \pi_{19}(d)$ jedes Element $ma_i \in MA_d$ auf dessen Konfigurationenelement $c_j \in C_d$ ab, welches die Konfiguration der von ma_i repräsentierten Managementaufgabe beschreibt:

$$konfiguration_d: MA_d \rightarrow C_d \cup \{\perp\}$$

Ist $c_j = konfiguration_d(ma_i)$, dann müssen die von c_j beschriebenen Konfigurationsinformationen bei der Ausführung der von ma_i repräsentierten Managementaufgabe berücksichtigt werden. Hat eine Aufgabe keine Konfiguration, wird dies durch $\perp$ abgebildet. ■

Die Abbildung $konfiguration_d$ ist dabei i. A. nicht surjektiv oder injektiv, da möglicherweise alle in einem DMMN-Modell beschriebenen Managementaufgaben eine Konfiguration haben und das Bildbereichselement $\perp$ in diesem Fall auf kein Annotationenelement abgebildet wird. Zudem kann es mehrere Annotationen geben, die keine Konfiguration haben und somit auf $\perp$ abgebildet werden. Die Menge aller Konfigurationenelemente aller DMMN-Modelle wird definiert als:

Definition 48 (Menge aller Konfigurationenelemente). Sei $D = \{d_1, \dots, d_n \mid n \in \mathbb{N}\}$ die Menge aller DMMN-Modelle, C_{d_i} die Menge aller Konfigurationenelemente eines DMMN-Modells $d_i \in D$, dann enthält die endliche Menge $C := \{c_1, c_2, \dots, c_n \mid n \in \mathbb{N}\}$ alle Konfigurationenelemente aller DMMN-Modelle $d_i \in D$. Diese wird als *Menge aller Konfigurationenelemente* bezeichnet:

$$C := \bigcup_{d_i \in D} C_{d_i}$$

■

4.2.4.2 Reihenfolgespezifikation

Deklarative Managementmodelle spezifizieren typischerweise keine Ausführungsreihenfolge der deklarativ modellierten Managementaufgaben (vgl. [Definition 11](#)). Diese wird prinzipiell vor deren Ausführung aus der Semantik der Aufgaben und deren kausalen Abhängigkeiten abgeleitet. Soll beispielsweise ein Webserver installiert und eine Anwendung darauf ausgeliefert werden, ergibt sich die Ausführungsreihenfolge der beiden Bereitstellungsaufgaben eindeutig aus deren Abhängigkeiten. Haben Managementaufgaben jedoch keine direkten Abhängigkeiten zueinander, woraus deren Ausführungsreihenfolge eindeutig ableitbar ist, kann dies zu ungewollten Resultaten führen. Sind für eine Datenbank beispielsweise zwei Managementaufgaben deklariert, um (i) Daten zu exportieren und (ii) Daten zu importieren, haben diese beiden Aufgaben keine direkten Abhängigkeiten: Beide Ausführungsreihenfolgen sind denkbar, beeinflussen jedoch die schlussendlich exportierten Daten. Die Ausführungsreihenfolge dieses Szenarios hängt somit von der Interpretation der Ausführungsumgebung ab und ist nicht eindeutig aus dem DMMN-Modell ableitbar. Um eine gewünschte Ausführungsreihenfolge jedoch explizit in einem DMMN-Modell zu modellieren, ermöglicht die DMMN-Klasse „Reihenfolgespezifikation“ die eindeutige Spezifikation, ob zwei Annotationen (i) gleichzeitig oder (ii) in einer bestimmten Reihenfolge ausgeführt werden müssen.

Definition 49 (Reihenfolgespezifikationenelemente). Sei $d \in D$ ein DMMN-Modell, dann enthält die endliche Menge $P_d := \pi_{11}(d) = \{p_1, p_2, \dots, p_n \mid n \in \mathbb{N}\}$ alle Instanzen p_i der DMMN-Klasse „Reihenfolgespezifikation“, welche jeweils spezifizieren, ob zwei Managementannotationen des DMMN-Modells d in einer bestimmten Reihenfolge ausgeführt werden müssen, oder ob der Effekt der Ausführung einer gleichzeitigen Ausführung der beiden Managementannotationen entsprechen muss. Diese Instanzen werden als *Reihenfolgespezifikationenelemente des Modells* bezeichnet. Jedes Element $p_i \in P_d$ ist dabei als Tripel definiert:

$$P_d \subseteq MA_d \times MA_d \times \{C, O, *\}$$
$$p_i = (ma_p, ma_a, Typ)$$

Dabei ist:

- $\pi_1(p_i) = ma_p, ma_p \in MA_d$, das Annotationenelement, dessen repräsentierte Managementaufgabe vor der von ma_a repräsentierten Managementaufgabe ausgeführt werden soll, falls $\pi_3(p_i) = O$,
- $\pi_2(p_i) = ma_a, ma_a \in MA_d$, das Annotationenelement, dessen repräsentierte Managementaufgabe nach der von ma_p repräsentierten Managementaufgabe ausgeführt werden soll, falls $\pi_3(p_i) = O$,
- $\pi_3(p_i) = Typ, Typ \in \{C, O, *\}$, der Typ der repräsentierten Reihenfolgespezifikation.

Falls $Typ = C$, dann muss der Effekt der Ausführung der von ma_p und ma_a repräsentierten Managementaufgaben einer gleichzeitigen Ausführung der beiden Aufgaben entsprechen. Falls $Typ = *$, dann sind explizit beide Ausführungsvarianten möglich, was semantisch dem Entfernen der Reihenfolgespezifikation entspricht. Diese *Wildcard* wird dazu verwendet, falsch positive bzw. vermeintliche Modellierungsfehler explizit auszuschließen. Es gilt die Eindeutigkeit der Ausführungsreihenfolge zweier Managementaufgaben:

$$\forall d \in D \forall p_1, p_2 \in P_d: \{\pi_1(p_1)\} \cup \{\pi_2(p_1)\} = \{\pi_1(p_2)\} \cup \{\pi_2(p_2)\} \Rightarrow p_1 = p_2$$

■

Die Menge aller Reihenfolgespezifikationenelemente aller DMMN-Modelle wird formal definiert wie folgt:

Definition 50 (Menge aller Reihenfolgespezifikationenelemente). Sei $D = \{d_1, \dots, d_n \mid n \in \mathbb{N}\}$ die Menge aller DMMN-Modelle, P_{d_i} die Menge aller Reihenfolgespezifikationenelemente eines DMMN-Modells $d_i \in D$, dann enthält die endliche Menge $P := \{p_1, p_2, \dots, p_n \mid n \in \mathbb{N}\}$ alle Reihenfolgespezifikationenelemente aller DMMN-Modelle $d_i \in D$. Diese wird als *Menge aller Reihenfolgespezifikationenelemente* bezeichnet:

$$P := \bigcup_{d_i \in D} P_{d_i}$$

■

4.3 Vordefinierte Relationen, Annotationen und Schnittstellen

Im Rahmen der DMMN-Spezifikation werden in diesem Abschnitt grundlegende Relationen, Annotationen und Managementschnittstellen definiert.

Zur Modellierung von Beziehungen werden die folgenden Relationentypen gemäß des *TOSCA Simple Profile* [OAS15] eingeführt. Sei $r_i \in R_d$ ein Relationenelement zwischen zwei Komponentenelementen $k_1, k_2 \in K_d$ in einem DMMN-Modell $d \in D$ und sei $\pi_3(r_i) = k_1$ und $\pi_4(r_i) = k_2$, dann gilt:

- Falls $\pi_2(\text{typ}_d(r_i)) = \text{HostedOn}$, dann wird die von k_1 repräsentierte Komponente auf der von k_2 repräsentierten Komponente betrieben.
- Falls $\pi_2(\text{typ}_d(r_i)) = \text{DependsOn}$, dann ist die von k_1 repräsentierte Komponente von der von k_2 repräsentierten Komponente abhängig.
- Falls $\pi_2(\text{typ}_d(r_i)) = \text{ConnectsTo}$, dann hat die von k_1 repräsentierte Komponente eine Verbindung zu der von k_2 repräsentierten Komponente.

Werden neue Relationentypen definiert, welche von diesen erben, wird die Semantik des jeweiligen Typs verfeinert aber in ihrer Grundbedeutung nicht geändert. Beispielsweise kann die ConnectsTo-Relation durch Vererbung zu einer „SQLConnection“ erweitert werden, um die Verbindung zwischen einer Anwendung und einer SQL-Datenbank detaillierter zu beschreiben.

Zur Deklaration der Bereitstellung von Komponenten und Relationen sowie deren Außerbetriebnahme werden folgende Managementannotationen eingeführt. Sei $te_i \in TE_d$ ein Topologieelement in einem DMMN-Modell $d \in D$ und $ma \in \text{annotationen}_d(te_i)$, dann gilt folgende Semantik:

- *Bereitstellungsannotation*: Falls $\pi_2(ma) = \text{Create}$, dann ist die von te_i repräsentierte Komponente bzw. Relation gemäß den Werten der Eigenschaftenelemente bereitzustellen, welche sich nach Auflösung der Vererbungsgesetze für te_i ergeben. Dabei müssen ausschließlich Eigenschaften berücksichtigt werden, die vom Typ und den Supertypen der von te_i repräsentierten Komponente bzw. Relation definiert sind. Nach Ausführung von ma gilt: $\exists e_j \in \text{eigenschaften}(te_i): \pi_2(e_j) = \text{State} \wedge \pi_3(e_j) = \text{Running}$.

- *Außerbetriebnahmeannotation*: Falls $\pi_2(ma) = Destroy$, dann ist die von te_i repräsentierte Komponente bzw. Relation außer Betrieb zu nehmen. Nach Ausführung von ma gilt:

$$\exists e_j \in eigenschaften(te_i): \pi_2(e_j) = State \wedge \pi_3(e_j) = Uninstalled.$$

Konfigurationen werden für diese Annotationstypen nicht vordefiniert, da diese von den jeweiligen Topologieelementen abhängig sind.

Um individuelle Komponenten und Relationen generisch in bzw. außer Betrieb zu nehmen, wird die *Lebenszyklusschnittstelle* (engl. „Lifecycle Interface“) gemäß des TOSCA-Primer [OAS13a] definiert. Diese Schnittstelle definiert fünf Operationstypen, die aufgrund der eindeutig beschriebenen Semantik zur generischen Bereitstellung und Außerbetriebnahme von Komponenten bzw. Relationen aufgerufen werden können. Dadurch kann die Ausführung von Managementannotationen bezüglich dieser Managementaufgaben beeinflusst und individuelle Managementlogik vom Modellierer spezifiziert werden. Sei $te_i \in TE_d$ ein Komponentenelement oder Relationenelement in einem DMMN-Modell $d \in D$ und $i_j \in schnittstellen(te_i)$ ein Schnittstellenelement des Typs $\pi_2(i_j) = LifecycleManagement$, dann gilt für die von $o_k \in operationen_d(i_j)$ repräsentierte Managementoperation folgende Semantik:

- Falls $\pi_2(o_k) = Install$ und $\exists e_j \in eigenschaften(te_i): \pi_2(e_j) = State \wedge \pi_3(e_j) = Uninstalled$, dann kann die von te_i repräsentierte Komponente bzw. Relation über die von o_k repräsentierte Operation installiert werden. Nach Ausführung der Operation gilt:

$$\exists e_j \in eigenschaften(te_i): \pi_2(e_j) = State \wedge \pi_3(e_j) = Installed$$
- Falls $\pi_2(o_k) = Configure$ und $\exists e_j \in eigenschaften(te_i): \pi_2(e_j) = State \wedge ((\pi_3(e_j) = Installed) \vee (\pi_3(e_j) = Configured))$, dann kann die von te_i repräsentierte Komponente bzw. Relation über die von o_k repräsentierte Operation konfiguriert werden. Nach Ausführung der Operation gilt:

$$\exists e_j \in eigenschaften(te_i): \pi_2(e_j) = State \wedge \pi_3(e_j) = Configured.$$
- Falls $\pi_2(o_k) = Start$ und die Komponente bzw. Relation te_i konfiguriert ist, d. h. $\exists e_j \in eigenschaften(te_i): \pi_2(e_j) = State \wedge \pi_3(e_j) = Configured$,

dann kann die von te_i repräsentierte Komponente bzw. Relation über die von o_k repräsentierte Operation o_k gestartet werden. Anschließend gilt:
 $\exists e_j \in \text{eigenschaften}(te_i): \pi_2(e_j) = \text{State} \wedge \pi_3(e_j) = \text{Running}$.

- Falls $\pi_2(o_k) = \text{Stop}$ und die Komponente bzw. Relation te_i in Betrieb ist, d. h. $\exists e_j \in \text{eigenschaften}(te_i): \pi_2(e_j) = \text{State} \wedge \pi_3(e_j) = \text{Running}$, dann kann die von te_i repräsentierte Komponente bzw. Relation über die von o_k repräsentierte Operation o_k gestoppt werden. Anschließend gilt:
 $\exists e_j \in \text{eigenschaften}(te_i): \pi_2(e_j) = \text{State} \wedge \pi_3(e_j) = \text{Configured}$.
- Falls $\pi_2(o_k) = \text{Uninstall}$ und die Komponente bzw. Relation installiert oder konfiguriert ist, das heißt $\exists e_j \in \text{eigenschaften}(te_i): \pi_2(e_j) = \text{State} \wedge (\pi_3(e_j) = \text{Installed} \vee \pi_3(e_j) = \text{Configured})$, dann kann die von te_i repräsentierte Komponente bzw. Relation über die von o_k repräsentierte Operation deinstalliert werden. Nach Ausführung der Operation gilt:
 $\exists e_j \in \text{eigenschaften}(te_i): \pi_2(e_j) = \text{State} \wedge \pi_3(e_j) = \text{Uninstalled}$.

Um die gesamte Inbetriebnahme einer Komponente oder Relation durch Ausführung einer einzigen Operation zu realisieren, wird folgende Semantik definiert:

- Falls $\pi_2(o_k) = \text{Install+}$ und $\exists e_j \in \text{eigenschaften}(te_i): \pi_2(e_j) = \text{State} \wedge \pi_3(e_j) = \text{Uninstalled}$, dann kann die von te_i repräsentierte Komponente bzw. Relation über die von o_k repräsentierte Operation installiert, konfiguriert und gestartet werden. Nach Ausführung der Operation gilt:
 $\exists e_j \in \text{eigenschaften}(te_i): \pi_2(e_j) = \text{State} \wedge \pi_3(e_j) = \text{Running}$.

Die Implementierungen der Operationen werden nicht von der Definition der Managementschnittstelle vorgeschrieben, wodurch dazu beliebige Technologien eingesetzt werden können. Entsprechend dem TOSCA-Primer wird für die Schnittstelle ebenso lediglich die Semantik der Operationstypen definiert, jedoch keine Ein- bzw. Ausgabeparameter, da diese von den jeweiligen Komponenten und Relationen abhängig sind, welche die Schnittstelle implementieren. Beim Aufruf der Operationen muss (i) die entsprechende Eingabeparametrisierung somit vom Aufrufenden aus dem Kontext abgeleitet werden. Um die Eingabeparametrisierung durch den Modellierer zu beeinflussen, können (ii) Eigenschaftsreferenzen für Parameter spezifiziert werden, die auf Eigenschaften

von Topologieelementen im DMMN-Modell zeigen (siehe [Definition 35](#)). Beim Aufruf der Operation müssen folglich die referenzierten Eigenschaften aus dem gegenwärtigen Instanzmodell abgerufen und als Eingabeparametrisierung verwendet werden. In gleicher Weise kann die Übertragung der Werte von Ausgabeparametern ins Instanzmodell spezifiziert werden. Die Definition dieser Schnittstelle erhebt keinen Anspruch auf Vollständigkeit und kann durch die Änderung der existierenden Zustandsdefinitionen modifiziert sowie durch das Hinzufügen weiterer Zustandsdefinitionen erweitert werden.

4.4 Grafische Notation

In diesem Abschnitt wird das Metamodell einer grafischen Notation für DMMN vorgestellt, welches die visuellen Modellierungskonstrukte für deklarative Managementmodelle definiert. Die Notation basiert auf VINO4TOSCA [BBK+12] und soll den Modellierer bei der Erstellung von DMMN-Modellen unterstützen und den Modellierungsprozess vereinfachen. Generell erleichtern grafische Notationen sowohl die Kommunikation als auch die Analyse von Modellen, da die visuellen Wahrnehmungsfähigkeiten des Menschen unterstützend einbezogen werden. Um eine kognitiv effektive grafische Notation zu entwickeln, wurden die Prinzipien von Moody [Moo09] angewandt, welche die physikalischen Wahrnehmungsfähigkeiten des Menschen explizit berücksichtigen und dadurch zur Optimierung der Lesbarkeit und Kommunikation grafischer Modelle beitragen. Die Notation dient als Grundlage für das DMMN-Modellierungswerkzeug.

In den folgenden Abschnitten wird die grafische Repräsentation aller drei Schichten des Metamodells vorgestellt. Für alle definierten Formen gilt: Linien- und Hintergrundfarbe sind frei wählbar, die Linienart der Formen ist jedoch semantisch belegt und muss getreu des definierten Schemas umgesetzt werden.

4.4.1 Darstellung der Topologie- und technischen Managementschicht

Die Notation spezifiziert die Darstellung folgender Modellelemente der Topologie- und technischen Managementschicht: Komponenten, Relationen, Management-schnittstellen, Operationen, Anwendungsartefakte und Eigenschaften.

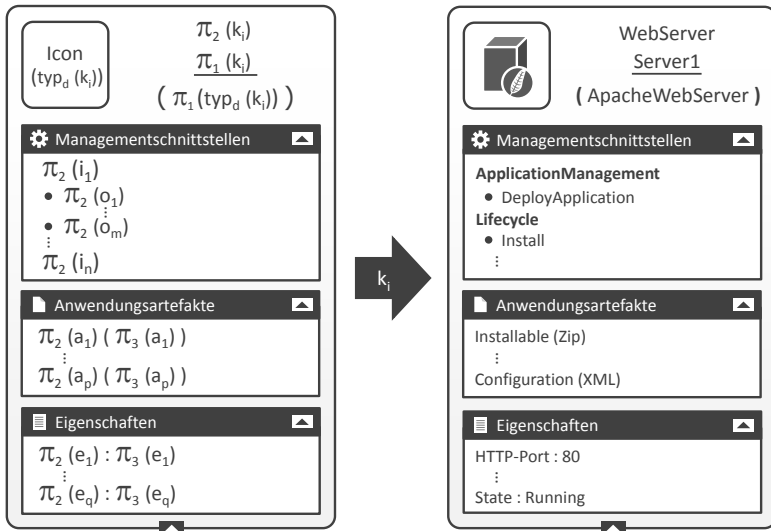


Abbildung 4.3: Darstellung eines Komponentenelements

4.4.1.1 Darstellung von Komponenten

Instanzen der Klasse „Komponente“ werden wie in [Abbildung 4.3](#) gezeigt als abgerundete Rechtecke dargestellt. Die linke Seite zeigt den schematischen Aufbau des grafischen Elements, auf der rechten Seite wird die Repräsentation einer realen Webserver-Komponente $k_i \in K_d$ eines Modells $d \in D$ dargestellt. Deren Attribut „Name“ wird als normaler Text dargestellt, das Attribut „Id“ als unterstrichener Text und der Komponententyp als eingeklammerter Text. Die Attribute sind optional und mittig in Aufzählungsreihenfolge angeordnet. Ein Icon kann verwendet werden, um den Komponententyp zu visualisieren.

Spezifiziert die darzustellende Komponente oder deren Supertypen Managementschnittstellen, werden diese innerhalb einer gleichnamigen Box als Liste deren Repräsentationen $\{i_1, \dots, i_n\} \in schnittstellen(k_i)$ abgebildet und durch das jeweilige „Typ“-Attribut typisiert. Die Namen der jeweiligen Operationenelemente $\{o_1, \dots, o_m\} \in operationen_d(i_j)$ einer von i_j repräsentierten Managementschnittstelle werden unter dieser in eingerückter Form aufgelistet.

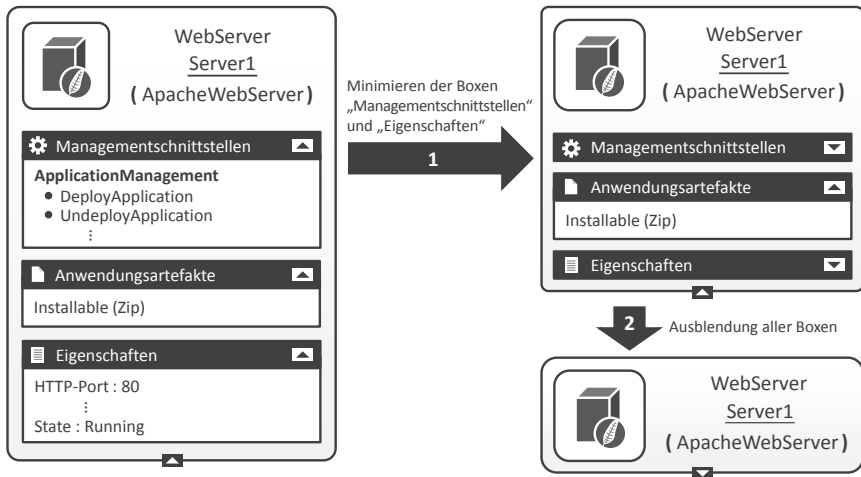


Abbildung 4.4: Alternative Darstellungsformen für Komponentenelemente

Anwendungsartefakte zur Bereitstellung der Komponente sind ebenfalls in einer gleichnamigen Box dargestellt. Darin werden alle Repräsentationen der Anwendungsartefakte $\{a_1, \dots, a_p\} \in \text{artefakte}(k_i)$ aufgelistet und jeweils der „Name“ des Artefakts dargestellt sowie dessen „Typ“ in Klammern abgebildet.

Die Eigenschaftenelemente $\{e_1, \dots, e_q\} \in \text{eigenschaften}(k_i)$ werden innerhalb einer Box dargestellt, welche jeweils das Attribut „Name“ sowie den derzeitigen bzw. gewünschten „Wert“ jeder Eigenschaft auflistet.

Die Boxen der Managementschnittstellen, Anwendungsartefakte und Eigenschaften können wie in [Abbildung 4.4](#) gezeigt unabhängig voneinander in minimierter bzw. maximierter Form dargestellt werden, um darin enthaltene Informationen grafisch ein- oder auszublenden (siehe Übergangspfeil 1). Zudem können alle Boxen komplett ausgeblendet werden, um eine kompakte Darstellung der Komponente zu erhalten (siehe Übergangspfeil 2). Bei der Implementierung eines DMN-Modellierungswerkzeugs können die jeweiligen Pfeil-Symbole der Boxen und die des abgerundeten Rechtecks zur Minimierung bzw. Maximierung der Darstellungsform als Button verwendet werden.

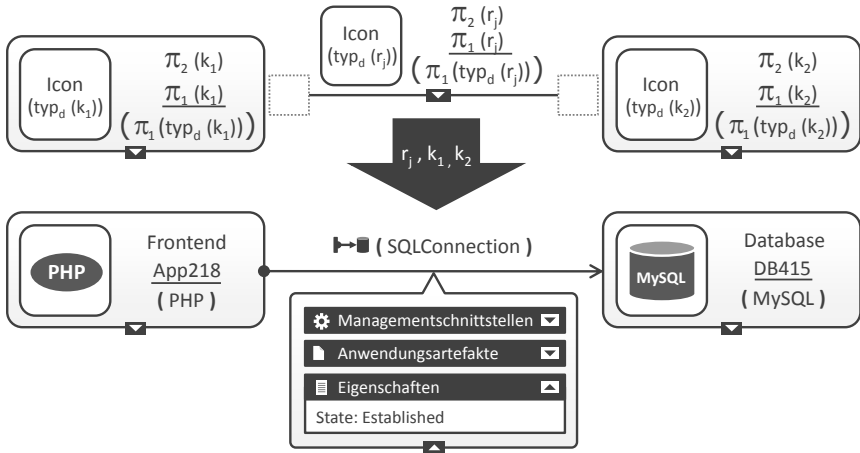


Abbildung 4.5: Darstellung eines Relationenelements

4.4.1.2 Darstellung von Relationen

Relationenelemente werden wie in [Abbildung 4.5](#) gezeigt als Pfeile zwischen Komponentenelementen dargestellt. Die obere Abbildungshälfte zeigt den schematischen Aufbau des grafischen Elements, während darunter ein konkretes Relationenelement $r_j \in R_d$ des Typs *SQLConnection* abgebildet wird, welches zwei durch $k_1, k_2 \in K_d$ repräsentierte Komponenten eines Modells $d \in D$ verbindet. Die beiden Enden eines Relationenpfeils können in Abhängigkeit des Relationentyps beliebig gestaltet werden. Die Darstellung der Attribute „Name“ und „Id“, des Typs sowie des Icons entspricht der Darstellung von Komponentenelementen. Diese werden bei horizontaler Ausrichtung oberhalb der Pfeillinie, bei vertikaler Ausrichtung links oder rechts der Pfeillinie abgebildet. Die Darstellung der Schnittstellen, Anwendungsartefakte und Eigenschaften entspricht der Darstellungsform für Komponenten. Die jeweiligen Boxen werden bei horizontaler Pfeillinie unterhalb, bei vertikaler Pfeillinie links oder rechts der Relation in einer rechteckigen Gruppierungsbox abgebildet. Diese kann minimiert (obere Hälfte in [Abbildung 4.5](#)) sowie maximiert (untere Hälfte in [Abbildung 4.5](#)) dargestellt werden, um Informationen ein- bzw. auszublenden.

Bereitstellungsannotation annotiert ist, durch gepunktete Linien grafisch markiert werden. Ein Beispiel dafür ist die Punktierung der SQL-Verbindung in [Abbildung 4.6](#), welche zu erzeugen ist und somit den Zielzustand der Relation beschreibt jedoch zum Modellierungszeitpunkt real nicht existiert. Formal kann diese grafische Unterscheidung wie folgt definiert werden:

Definition 51 (Linienpunktierung grafischer Elemente). Sei $te_i \in TE_d$ ein Topologieelement in einem DMMN-Modell $d \in D$, dann gilt folgende Beziehung für die Liniendarstellung von te_i :

Linie durchgezogen $:\Leftrightarrow$

$$\nexists e_j \in \text{eigenschaften}(te_i) : \pi_2(e_j) = \text{State} \wedge \pi_3(e_j) = \text{Uninstalled}$$

Linie punktiert $:\Leftrightarrow$

$$\exists e_j \in \text{eigenschaften}(te_i) : \pi_2(e_j) = \text{State} \wedge \pi_3(e_j) = \text{Uninstalled}$$

■

Durch die retinale visuelle Variable der Textur [[Moo09](#)] ermöglicht dieser grafische Repräsentationsunterschied die visuelle Unterscheidung real existierender Komponenten und Relationen von denen, die nicht existieren und folglich zu erzeugen sind. Die DMMN-Klassen „Konfiguration“ und „Reihenfolgespezifikation“ werden nicht durch eigene grafische Elemente abgebildet.

4.4.3 Beispiel eines DMMN-Modells

Basierend auf der im vorherigen Abschnitt eingeführten grafischen Notation wird in diesem Abschnitt ein Beispiel beschrieben, welches die Verwendung und das Zusammenspiel der verschiedenen Formen anhand eines realen Anwendungsfalls illustriert. In [Abbildung 4.7](#) wird ein DMMN-Modell grafisch mithilfe der definierten Notation modelliert. Das Modell beschreibt auf der rechten Seite eine existierende MySQL-Datenbankinfrastruktur, welche auf dem Cloud-Dienst Amazon Elastic Compute Cloud (Amazon EC2) in Form einer virtuellen Linux-Maschine der Ubuntu-Distribution betrieben wird. Die vier repräsentier-

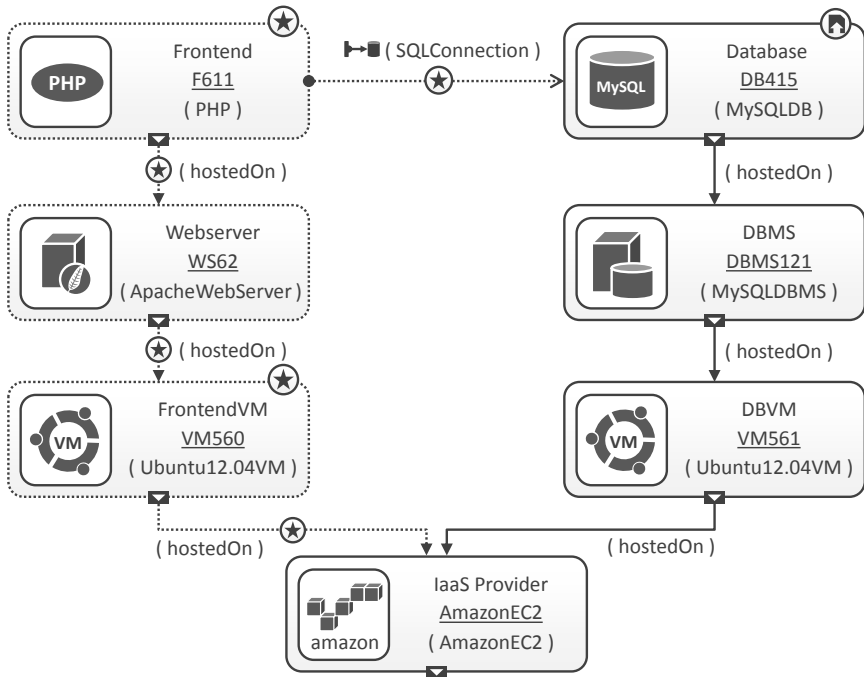


Abbildung 4.7: Beispiel eines DMMN-Modells

ten Komponenten sind jeweils durch eine Relation des Typs *hostedOn* mit der darunterliegenden Komponente verbunden, auf welcher sie ausgeführt werden. Um die reale Existenz dieser Komponenten und Relationen zu kennzeichnen, sind die Linien der jeweiligen Formen durchgezogen. An der Datenbank ist zudem eine Managementannotation zum Import von Daten deklariert.

Auf der linken Seite wird präskriptiv die Infrastruktur zum Betrieb einer PHP-Anwendung auf einem Apache-Webserver beschrieben, welche ebenfalls auf einer virtuellen Linux-Maschine selben Distributionstyps basiert, jedoch zum Modellierungszeitpunkt nicht existiert: Alle Komponenten- und Relationenelemente dieses Infrastrukturzweigs sind mit einer Bereitstellungsannotation annotiert, wodurch deren auszuführende Bereitstellung deklarativ modelliert wird. Daher sind die Komponenten- und Relationenelemente dieses Zweigs

durch Punktierung der Linien gekennzeichnet, wodurch die Nichtexistenz der repräsentierten Komponenten bzw. Relationen visualisiert wird. Zwischen den beiden Infrastrukturen ist eine Relation des Typs *SQLConnection* modelliert, welche ebenfalls durch eine Bereitstellungsannotation deren auszuführende Inbetriebnahme deklariert und die PHP-Anwendung mit der MySQL-Datenbank verbindet. Mögliche Eigenschaften, Artefakte und Managementschnittstellen der dargestellten Komponenten und Relationen sind in dieser Darstellung zur kompakteren Repräsentation ausgeblendet, können aber falls nötig mithilfe der zugehörigen Boxen beschrieben werden (siehe [Abschnitt 4.4.1](#)).

Die Semantik dieses DMMN-Modells kann somit wie folgt zusammengefasst werden: Eine PHP-Anwendung sowie deren Infrastruktur sollen auf Basis des Cloud-Dienstes Amazon EC2 provisioniert und mit einer existierenden MySQL-Datenbank verbunden werden, in welche Daten importiert werden sollen.

4.5 Serialisierung von DMMN-Modellen

Zur maschinenlesbaren Verarbeitung von DMMN-Modellen werden in diesem Abschnitt deren Serialisierungsmöglichkeiten diskutiert. Grundsätzlich erlaubt das DMMN-Metamodell jegliche Serialisierungsform, welche die involvierten graphbasierten Konzepte abbilden kann. Eine wesentliche Anforderung daran ist jedoch aufgrund der nicht trivialen Klassenstruktur die Möglichkeit das Metamodell so abzubilden, dass es sich strukturiert erfassen lässt und zur automatisierten Instanziierung und Validierung von DMMN-Modellen herangezogen werden kann. Viele Sprachen zur Modellierung von Architekturen und Managementbeschreibungen setzen dazu die *Extensible Markup Language (XML)* ein, welche die oben genannte Anforderung in Form der *Extensible Schema Definition (XSD)* erfüllt. Der Prototyp des PALMA-Frameworks unterstützt daher eine XML-basierte Sprache zur Modellierung von DMMN-Modellen, deren Metamodell mittels XSD modelliert ist. Aufgrund der vergleichsweise hohen Anzahl an Zeichen, die zur Beschreibung von XML-Modellen erforderlich sind, unterstützt der Prototyp zusätzlich DMMN-Repräsentationen in der *JavaScript Object Notation (JSON)* und der *YAML Ain't Markup Language (YAML)*.

4.6 Diskussion

Ein wesentlicher Unterschied von DMMN zu anderen deklarativen Modellierungssprachen des Anwendungsmanagements ist die geschaffene Integration unterschiedlicher Modellierungsschichten. Durch die deskriptive Beschreibung der Anwendung sowie der präskriptiven deklarativen Beschreibung auszuführender Managementaufgaben, der Zielstruktur und des Zielzustandes der Anwendung, wird so ein ganzheitliches, integriertes Metamodell geschaffen, welches sowohl zur Modellierung zustandsmodifizierender als auch zustandserhaltender Managementaufgaben eingesetzt werden kann (vgl. [Abschnitt 2.3.1](#)). Das DMMN-Metamodell erfüllt somit alle Anforderungen, welche im Rahmen der PALMA-Methode an deklarative Managementmodelle definiert wurden (vgl. [Definition 11](#)). Die Abbildung des gesamten Ausführungskontextes von Managementaufgaben ermöglicht zudem die Analyse der Modelle hinsichtlich enthaltener Managementfehler, wodurch die Grundlage für die kontextsensitive Analyse von Managementaufgaben ([Forschungsbeitrag 5](#)) gelegt wird. Zielzustandsmodelle sind somit konzeptionell als echte Teilmenge der DMMN-Modelle zu sehen, da DMMN den Inhalt dieser Modelle generell beschreiben kann, jedoch zusätzlich zustandserhaltende Managementaufgaben unterstützt.

Bei der Entwicklung von DMMN wurden Klassen und Konzepte existierender Metamodelle wiederverwendet und adaptiert, die deklarative Management-schicht wird allerdings von keinem der existierenden Metamodelle berücksichtigt und stellt somit eine paradigmatische Erweiterung dar. Die Repräsentation der Topologieschicht und technischen Managementschicht ist vom TOSCA-Standard [[OAS13b](#); [OAS15](#)] inspiriert und adaptiert einige Konzepte zur Erhöhung der Flexibilität: DMMN erlaubt eine flexiblere Individualisierung der Topologieelemente durch die Möglichkeit der Definition von Managementschnittstellen, Artefakten und Eigenschaften direkt am Topologieelement - unabhängig von dessen Typ oder Supertypen. Ähnlich wie in modernen Programmiersprachen können Managementschnittstellen dadurch direkt auf Objektebene, d. h. auf Komponenten- bzw. Relationenebene, spezifiziert werden. Damit überträgt DMMN das Konzept der *Interfaces* aus objektorientierten Programmiersprachen

auf Managementschnittstellen und ermöglicht Modellierenden die Modelle flexibel manuell zu konfigurieren bzw. zu individualisieren. Dadurch können Topologieelemente individualisiert werden, ohne künstliche Komponenten- bzw. Relationentypen einführen zu müssen, die ausschließlich zur Definition einer bestimmten Schnittstelle bzw. deren Implementierung erstellt werden. Des Weiteren wird die Modellierung aufgrund der Reduktion von künstlichen Typhierarchien deutlich flexibler und führt zu übersichtlicheren Modellen. Die grundlegenden Konzepte von DMMN bezüglich der graphbasierten Topologiebeschreibung, der Art und Weise Eigenschaften, Managementschnittstellen, Artefakte und Operationen zu spezifizieren, sind daher von TOSCA inspiriert, wurden jedoch im Rahmen dieser Arbeit hinsichtlich ihrer Flexibilität gesteigert. Aus Sicht von DMMN ermöglicht das TOSCA-Metamodell somit prinzipiell die Modellierung einer restriktiven Form der Topologieschicht und der technischen Managementschicht, jedoch nicht der deklarativen Managementschicht.

Das DMMN-Metamodell nutzt zur Beschreibung der Instanzinformationen ähnliche Klassen wie das ETG-Metamodell [BFL+12; Bin15], erweitert dieses jedoch um Datentypen von Eigenschaften. Das *Common Information Model (CIM)* [DMT10] ist ein Standard von der *Distributed Management Task Force (DMTF)* für das Management von IT-Systemen. CIM definiert ein Metamodell, um die Managementinformationen der Hard- und Softwarekomponenten eines Systems sowie deren aufrufbare Operationen zu beschreiben. DMMN verwendet zur Beschreibung aufrufbarer Operationen an Komponenten sowie der Darstellung von Laufzeitinformationen einen ähnlichen Ansatz.

Die grafische Notation von DMMN basiert aufgrund der grundsätzlichen Topologieausrichtung auf VINO4TOSCA [BBK+12], erweitert diese jedoch um die Repräsentation von Instanzinformationen, Schnittstellen, Artefakten und der deklarativen Managementschicht. Andere Notationen, wie die FMC [KW03], UML 2.0 Komponentendiagramme [OMG07] oder ArchiMate [HP12], können prinzipiell gleichermaßen um die fehlenden Konzepte erweitert werden. In Breitenbücher et al. [BBK+13b; BBF+14] wurde zudem vorgestellt, wie nicht-funktionale Anforderungen an die Ausführung deklarativ modellierter Managementannotationen in DMMN-Modellen spezifiziert werden können.

KAPITEL 5

GENERIERUNG VON MANAGEMENTPLÄNEN

Mit dem DMMN-Metamodell wurde im vorangegangenen Kapitel eine Sprache zur Spezifikation deklarativer Managementmodelle vorgestellt, welche gemäß [Abschnitt 3.3.3](#) nicht direkt ausführbar sind: Deklarative Managementmodelle beschreiben ausschließlich *welche* Managementziele erreicht werden sollen, jedoch ohne zu spezifizieren, *wie* dies technisch durchzuführen ist. Die PALMA-Methode definiert daher einen Schritt zur Transformation von deklarativen Managementmodellen in ausführbare imperative Managementmodelle.

In diesem Kapitel wird ein Transformationsverfahren vorgestellt, welches diesen Schritt der Methode automatisiert. Wiederverwendbare Managementbausteine namens *Management-Planlets* werden zur Generierung eines ausführbaren imperativen Managementmodells orchestriert, welches auf Basis der Workflow-Technologie realisiert wird und alle in einem DMMN-Modell deklarierten Managementannotationen auf der realen Anwendung ausführt. Das Transformationsverfahren stellt [Forschungsbeitrag 3](#) dieser Arbeit dar.

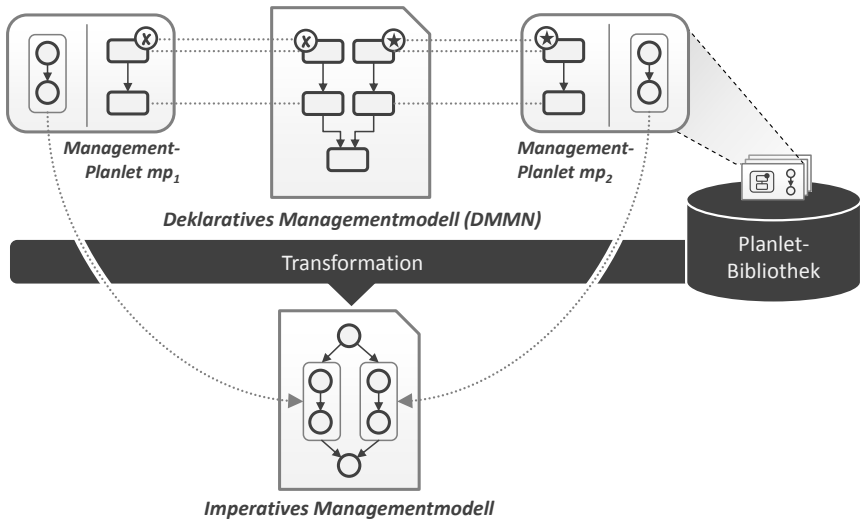


Abbildung 5.1: Grundkonzept der Transformation deklarativer Managementmodelle in ausführbare imperative Managementmodelle

5.1 Idee und Grundkonzept

Die zentrale Idee dieses Forschungsbeitrags ist die vollautomatisierte Orchestrierung wiederverwendbarer Managementsubprozesse mithilfe einer formalen Beschreibung ihrer Managementfunktionalität, um deklarative Managementmodelle auf Basis von DMMN in imperative Managementmodelle zu transformieren. Eine Darstellung dieser Idee wird in [Abbildung 5.1](#) gezeigt, worin das im nachfolgenden Kapitel eingeführte Konzept der Management-Planlets zur Formalisierung von Managementsubprozessen verwendet wird. Bei dieser Transformation wird für jede deklarativ beschriebene Managementaufgabe ein geeignetes Management-Planlet eingebunden, das die deklarierte Aufgabe ausführt. Dabei ist die Untersuchung, auf welcher Granularitätsebene imperative Managementmodelle von individuellen Anwendungen entkoppelt und als eigenständiges Management-Planlet zur Verfügung gestellt werden können, von zentraler wissenschaftlicher Bedeutung und grundlegend für diesen Beitrag.

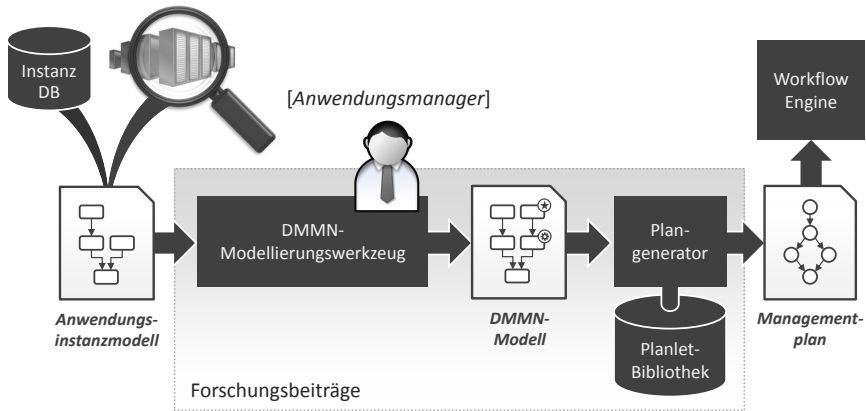


Abbildung 5.2: Architektur des Management-Planlet Frameworks

Während sich aufgrund der Anforderungen aus [Abschnitt 3.2](#) ausschließlich DMMN zur Modellierung deklarativer Managementmodelle eignet, können imperative Managementmodelle und Management-Planlets generell durch beliebige imperative Technologien umgesetzt werden, beispielsweise als Skripte oder Workflows. Die Konzepte dieses Forschungsbeitrags werden daher generell unabhängig von konkreten imperativen Technologien definiert. Eine spezifische Umsetzung imperativer Managementmodelle auf Basis der Workflow-Technologie, d. h. als Managementpläne, wird in [Abschnitt 5.5](#) vorgestellt, um die praktische Realisierbarkeit der theoretischen Konzepte zu validieren.

[Abbildung 5.2](#) zeigt die Kombination des vorherigen Forschungsbeitrags, der deklarativen Managementbeschreibungssprache DMMN, mit diesem Forschungsbeitrag in Form der Architektur des resultierenden Management-Frameworks, welches in dieser Variante als *Management-Planlet Framework* bezeichnet wird. Die Konzeption ermöglicht die Integration beider Managementparadigmen: Der Anwendungsmanager spezifiziert Managementziele deklarativ auf hoher Abstraktionsebene durch Modellierung von DMMN-Modellen, welche anschließend vollautomatisiert mittels eines Plangenerators durch Orchestrierung von Management-Planlets in ausführbare Managementpläne transformiert werden.

5.2 Management-Planlets

In diesem Abschnitt wird das Konzept der Management-Planlets eingeführt, wodurch Managementprozesse als (i) ausführbare, (ii) wiederverwendbare und (iii) komponierbare Managementbausteine formalisiert werden können. Management-Planlets sind imperative Managementmodelle, die eine abgeschlossene Menge an Managementaufgaben auf einer definierten Topologiestruktur ausführen. Beispielsweise, um eine virtuelle Maschine bereitzustellen, einen Webserver auf einem Betriebssystem zu installieren oder Daten aus einer Datenbank zu exportieren. Der Begriff „*Management-Planlet*“¹ ist dabei vom englischen „*Management Plan*“ abgeleitet, welcher unter anderem durch den OASIS Standard TOSCA [OAS13b] definiert wurde. Im Gegensatz zu Managementplänen, die eine umfangreiche Managementfunktionalität modellieren und stark an individuelle Anwendungen gekoppelt sind (vgl. [Abschnitt 2.3.5.1](#)), realisiert ein Planlet eine kleinere, in sich abgeschlossene Menge zusammenhängender Managementaufgaben. Daher betreffen die Managementaufgaben eines Planlets nur eine kleine Teilstruktur von Anwendungen, wodurch die Kopplung deutlich reduziert und die Wiederverwendbarkeit signifikant erhöht wird. Management-Planlets sind daher wiederverwendbare, generische Subprozesse für Managementpläne und somit komponierbare *Teilpläne*, was durch das Suffix „let“ ausgedrückt wird. Die Ausführung eines Planlets ist dabei *atomar*, d. h. es wird entweder vollständig erfolgreich ausgeführt oder unter Rücksetzung aller durchgeführten Managementaufgaben terminiert (engl. „*rollback*“).

Definition 52 (Management-Planlet - informell). Ein Management-Planlet ist ein komponierbares imperatives Managementmodell, das ein oder mehrere Managementaufgaben abgeschlossen, d. h. atomar, auf einer bestimmten Topologiestruktur ausführt und sowohl (i) die Managementaufgaben, (ii) die Topologiestruktur, als auch (iii) die zur Ausführung geforderten Vorbedingungen sowie (iv) die Effekte der Ausführung formal spezifiziert. ■

¹Marrella et al. [MRM12] bezeichnen YAWL-Prozessmodelle als „Planlets“, die Vor- und Nachbedingungen sowie Effekte deren Aktivitäten deklarieren, um die automatische, dynamische Adaption einer Prozessinstanz im Falle auftretender Probleme mittels AI-Planning zu ermöglichen.

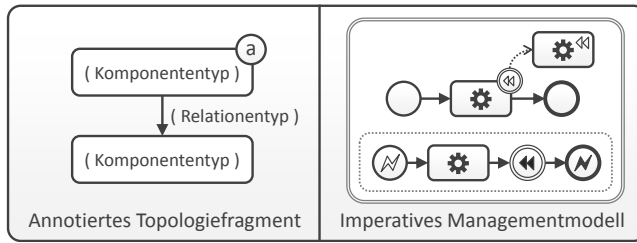


Abbildung 5.3: Konzeptioneller Aufbau eines Management-Planlets

5.2.1 Konzeptioneller Aufbau von Management-Planlets

Ein Planlet besteht aus einer (i) formalen Beschreibung seiner Funktionalität und einem (ii) imperativen Managementmodell, welches diese ausführt.

Zur formalen Beschreibung der Funktionalität eines Management-Planlets wird die Anwendungsstruktur, auf die das Planlet angewendet werden kann sowie die Managementaufgaben, welche das Planlet auf dieser ausführt, in Form eines DMMN-Modells beschrieben. Dieses wird als *annotiertes Topologiefragment* bezeichnet. Die darin modellierte Struktur, deren Eigenschaften, Schnittstellen und Artefakte definieren Vorbedingungen, die vom zu transformierenden DMMN-Modell erfüllt sein müssen, um das Planlet auszuführen. Ist das Fragment inklusive der Annotationen auf einen Teilgraphen eines DMMN-Modells abbildbar, kann das Planlet die übereinstimmenden Annotationen für die jeweilige Anwendung ausführen. Die Effekte der Ausführung werden durch separate Eigenschaften der Topologieelemente formalisiert (siehe [Abschnitt 5.2.5](#)).

Das imperative Managementmodell ist als *traditioneller Subprozess* [KELU10] modelliert und hat daher per Definition genau einen Anfang und ein Ende („SESE“). Treten bei dessen Ausführung nicht behandelbare Fehler auf, werden alle vom Planlet getätigten Änderungen sofort durch lokale Fehlerbehandlungs- und Kompensationsmethoden rückgängig gemacht und die Fehler an den Vaterprozess eskaliert. Zudem gibt ein Planlet die Autonomie der Lebensdauer zugunsten seines Vaterprozesses auf und kann jederzeit während seiner Ausführung, inklusive Rücksetzung aller getätigten Änderungen, terminiert werden.

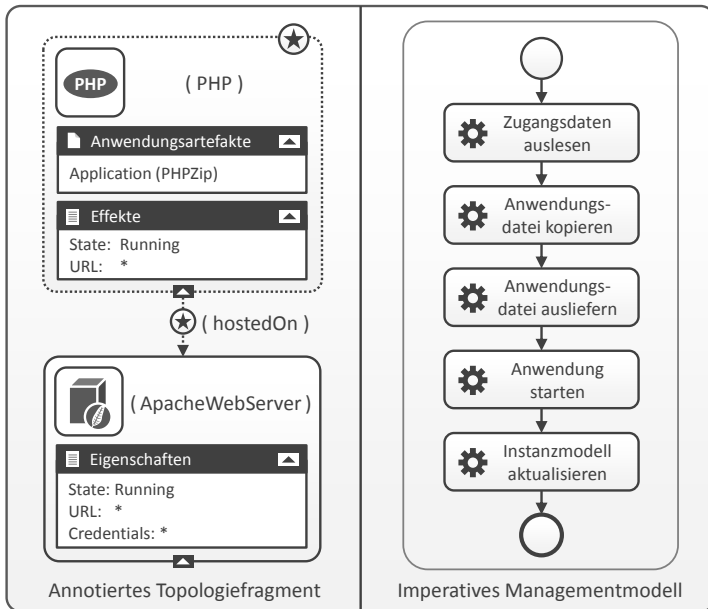


Abbildung 5.4: Beispiel eines Management-Planlets (Vereinfachte Darstellung ohne Fehlerbehandlungs- und Kompensationslogik)

Abbildung 5.4 erläutert dieses Konzept anhand eines konkreten Beispiels. Das gezeigte Planlet beschreibt durch das annotierte Topologiefragment, dass es eine PHP-Anwendung auf einem existierenden Apache Webserver installieren kann. Dazu sind sowohl am *PHP*-Komponentenelement als auch am ausgehenden *hostedOn*-Relationenelement jeweils eine Bereitstellungsannotation modelliert. Die Vorbedingungen dieses Planlets sind auf drei Arten ausgedrückt: (i) Anwendungsstruktur, (ii) Eigenschaften und (iii) Verfügbarkeit eines Artefakts. Das Planlet kann auf alle DMMN-Modelle angewendet werden, welche die spezifizierte Anwendungsstruktur als Subgraph enthalten. Dafür muss das DMMN-Modell jedoch auch deren Eigenschaften, welche vom Planlet als Vorbedingungen definiert sind, erfüllen: Die Komponente *ApacheWebServer* muss aktuell betrieben werden (*State*-Eigenschaft mit Wert „Running“) und zudem dessen Endpunkt (*URL*-Eigenschaft mit Wert „*“) sowie die Zugangsdaten zur

Administrationsschnittstelle (*Credentials*-Eigenschaft mit Wert „*“) spezifizieren. Das Sternsymbol drückt dabei einen beliebigen, nicht-leeren Wert aus. Als weitere Vorbedingung definiert das Planlet neben diesen Eigenschaften zusätzlich die Existenz eines Anwendungsartefakts der *PHP*-Komponente namens „Application“ des Typs „PHPZip“. Dieses Artefakt beinhaltet die Anwendungsdateien und wird vom imperativen Managementmodell des Planlets verwendet, um die Anwendung auf dem Webserver zu installieren. Die Effekte der Ausführung eines Management-Planlets werden grafisch durch eine separate Box repräsentiert, welche durch das Piktogramm der Eigenschaften die Zugehörigkeit zu dieser Klasse ausdrückt, sich jedoch durch ihre Beschriftung von diesen unterscheidet. Die Effekte des Planlets sind an der *PHP*-Komponente in Form der *State*- und *URL*-Eigenschaft spezifiziert, welche nach erfolgreicher Ausführung auf „Running“ bzw. einen noch unbekannten Wert gesetzt werden. Die Effekte beschreiben formal die erfolgreiche Bereitstellung der Komponente und werden ähnlich für die bereitzustellende *hostedOn*-Relation gesetzt (nicht dargestellt).

Auf der rechten Seite von [Abbildung 5.4](#) ist das imperative Managementmodell des Planlets graphisch in Form eines BPMN-Prozessmodells dargestellt. Die darin beschriebenen Schritte werden in Sequenz durchlaufen und installieren die Anwendung auf dem Webserver. Alle technischen Details der Installationsdurchführung sind durch die Aktivitäten des Prozesses implementiert. Das imperative Managementmodell kann dadurch beliebige (heterogene) Managementtechnologien integrieren, beispielsweise Management APIs aufrufen oder Skripte ausführen, ohne die technischen Details an den Aufrufenden zu exponieren: Die technische Ebene wird vollständig durch die technologieunabhängigen Managementannotationen, welche im Topologiefragment des Planlets deklariert sind, abstrahiert. Somit können Planlets auf Basis abstrakter, technologieunabhängiger Annotationen unabhängig der vom Planlet eingesetzten Managementtechnologien semantisch orchestriert werden, wodurch eine generische Integrationsschicht realisiert wird. BPMN wird in diesem Kapitel ausschließlich zur grafischen Modellierung imperativer Managementmodelle verwendet. Zu deren Umsetzung sind generell alle imperativen Technologien geeignet, welche die in [Abschnitt 5.2.4](#) erhobenen Anforderungen erfüllen.

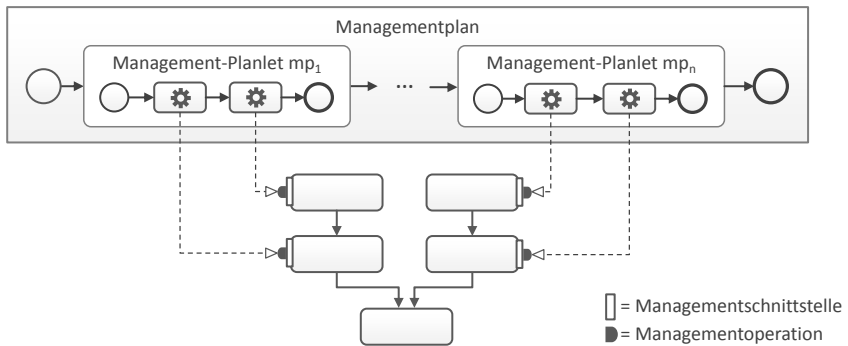


Abbildung 5.5: Verhältnis von Plänen, Planlets und Managementoperationen

5.2.2 Managementgranularität und Kopplung

Zur Ausführung von Managementaufgaben kann ein Management-Planlet beliebige Managementdienste, Programme oder Skripte aufrufen, welche entweder (i) als Managementoperationen im DMMN-Modell der betroffenen Anwendung beschrieben sind, (ii) vom Management-Planlet selbst verwaltet werden, z. B. durch eingebettete Skripte, oder (iii) aus dritten Quellen bezogen werden, beispielsweise durch den Aufruf des Management-API eines Cloud-Anbieters.

Das Verhältnis zwischen Plänen, Planlets und Operationen bezüglich der implementierten Managementgranularität und der Kopplungsstärke an individuelle Anwendungen wird in [Abbildung 5.5](#) verdeutlicht. Während Managementoperationen an genau einen Komponenten bzw. Relationentyp gebunden sind, rufen Planlets mehrere dieser Operationen auf, Pläne eine Vielzahl. Die Kopplung einer Operation an eine einzelne Anwendung ist daher minimal, da sie unabhängig von anderen Komponenten und Relationen aufgerufen werden kann. Planlets rufen typischerweise mehrere Operationen verschiedener Komponenten und Relationen auf, wodurch ein Planlet an eine bestimmte Struktur gekoppelt ist. Im Vergleich zu Managementplänen, die aufgrund zahlreicher Operationsaufrufe an große Teile einer Anwendung gekoppelt sind und typischerweise nur für einzelne Anwendungen funktionieren (vgl. [Abschnitt 2.3.5.1](#)), ist die Wiederverwendbarkeit eines Management-Planlets somit signifikant höher.

5.2.3 Informationsübertragung und Datenfluss

Zur Ausführung eines Planlets werden dessen imperativem Managementmodell folgende Eingaben übermittelt: (i) Ein Korrelationsbezeichner zur Identifikation der Ausführung, (ii) das DMMN-Modell, für welches das Planlet ausgeführt werden soll, (iii) eine Referenz zum Instanzmodell der Anwendung sowie (vi) eine Zuordnung der Elemente des annotierten Topologiefragments auf die Elemente des DMMN-Modells, wodurch auch die auszuführenden Annotationen zugewiesen werden. Durch diese Informationen kann ein Planlet Konfigurationen der auszuführenden Annotationen direkt aus dem DMMN-Modell auslesen und aktuelle Instanzinformationen über Dereferenzierung des Instanzmodells abrufen. Während der Ausführung überträgt das Planlet alle getätigten Änderungen unmittelbar ins Instanzmodell der Anwendung, wodurch dieses zu jedem Zeitpunkt deren gegenwärtigen Zustand repräsentiert. Die Übertragung der Eingaben hängt von der Umsetzung des Planlets als „*Inline Subprocess*“ oder „*Standalone Subprocess*“ ab [KKL+05], was in [Abschnitt 5.5.1](#) diskutiert wird.

Planlets tauschen Informationen implizit über das Instanzmodell der Anwendung aus, welches die einzige Kommunikationsform zwischen Planlets darstellt. Spezifiziert ein DMMN-Modell beispielsweise die Aufgaben zur Installation eines Datenbanksystems sowie der Erstellung einer Datenbank auf diesem, können die beiden Aufgaben durch zwei Planlets ausgeführt werden, welche den Endpunkt und die Zugangsdaten des Datenbanksystems über das Instanzmodell kommunizieren. Das imperative Managementmodell eines Planlets spezifiziert daher keinerlei Ausgabeparameter. Lese-Schreib-Konflikte bezüglich Eigenschaften werden durch temporale Ordnung der Planlets durch die Orchestrierung gelöst (siehe [Abschnitt 5.4](#)). Zudem ist jedes Planlet für die Dereferenzierung von Eigenschaftsreferenzen aufzurufender Operationen verantwortlich: Hat ein Planlet eine Operation als Vorbedingung, muss es beim Aufruf der entsprechenden Operation des DMMN-Modells die darin referenzierten Eigenschaften aus dem Instanzmodell lesen und zur Parametrisierung des Operationsaufrufes verwenden. Gleichmaßen müssen über Ausgabeparameter zurückgelieferte Werte nach Operationsausführung vom Management-Planlet ins Instanzmodell übertragen werden, falls für diese Eigenschaftsreferenzen spezifiziert sind.

5.2.4 Fehlerbehandlungs- und Kompensationsmechanismen

Planlets implementieren drei Mechanismen, deren Aufrufe die vom Planlet getätigten Änderungen zurücksetzen: (i) Vollständige Kompensation einer abgeschlossenen Planlet-Ausführung, (ii) Abbruch und Rücksetzung aller Änderungen durch ein laufendes Planlet selbst aufgrund eines Fehlers und (iii) Terminierung und Rücksetzung aller Änderungen im Auftrag des Vaterprozesses. Die zur Umsetzung imperativer Managementmodelle eingesetzte Technologie muss daher die Realisierung dieser Mechanismen ermöglichen. In [Abschnitt 5.5.2](#) wird eine mögliche Realisierung für die Workflow-Sprache BPEL vorgestellt.

Im ersten Fall ist die Ausführung eines Planlets innerhalb des Vaterprozesses erfolgreich abgeschlossen. Tritt zu einem späteren Zeitpunkt bei einem anderen Planlet ein nicht von diesem selbst behandelbarer Fehler auf, kann der Gesamtprozess nicht erfolgreich ausgeführt werden und muss abgebrochen werden. Um die Anwendung nicht in einen inkonsistenten Zustand zu führen, muss sie in ihren Ursprungszustand zurückgesetzt und alle bisher durchgeführten Aufgaben rückgängig gemacht werden: Sowohl das fehlerhaft ausgeführte Planlet (siehe nächster Paragraph) als auch alle zu diesem Zeitpunkt erfolgreich ausgeführten Planlets und noch laufenden Planlets müssen getätigte Änderungen rückgängig machen und terminieren. Jedes Planlet spezifiziert dazu eine Kompensationslogik, welche vom Vaterprozess aufgerufen werden kann, um eine erfolgreich abgeschlossene Planlet-Ausführung rückgängig zu machen.

Tritt während einer Planlet-Ausführung ein lokaler, nicht vom Planlet selbst behandelbarer Fehler auf, führt das Management-Planlet automatisch Fehlerbehandlungs- und Kompensationsmechanismen aus, um alle ausgeführten Aufgaben rückgängig zu machen und die eigene Ausführung abubrechen. Zudem eskaliert das Planlet den Fehler an den Vaterprozess, welcher die Terminierung bzw. Kompensation aller anderen Planlets innerhalb des Prozesses veranlasst.

Um laufende Management-Planlets innerhalb eines Vaterprozesses zu terminieren, stellt jedes Planlet einen Terminierungsmechanismus zur Verfügung. Dieser kann vom Vaterprozess aufgerufen werden, um alle bisher vom Planlet getätigten Änderungen rückgängig zu machen und die Ausführung abubrechen. Alle Mechanismen reflektieren durchgeführte Änderungen direkt im Instanzmodell.

5.2.5 Formales Metamodell

In diesem Abschnitt wird das formale Metamodell für Planlets vorgestellt.

Definition 53 (Management-Planlet). Ein Management-Planlet ist ein imperatives Managementmodell, dessen Anwendbarkeit und Funktionalität durch ein formales DMMN-Modell beschrieben ist. Die Menge aller Planlets ist definiert als $MP := \{mp_1, \dots, mp_n \mid n \in \mathbb{N}\}$. Jedes Planlet $mp \in MP$ ist ein Quadrupel:

$$MP \subseteq D \times \wp(EF) \times F \times IM$$

$$mp = (tf_{mp}, EF_{mp}, effekte_{mp}, im_{mp})$$

Dabei ist:

- $\pi_1(mp) = tf_{mp}, tf_{mp} \in D$, das annotierte DMMN-Topologiefragment, das sowohl (i) die Anwendbarkeit des Planlets als auch (ii) die ausführbaren Managementannotationen formal beschreibt. Der Graph des Topologiefragments ist dabei mindestens schwach zusammenhängend.
- $EF := \Sigma^+ \times \Sigma^+ \times (\Sigma^* \cup \{\perp, *\}) \times \Sigma^+$ die Menge aller Effektelemente, die syntaktisch äquivalent zu Eigenschaftenelementen aufgebaut sind und deren Semantik bezüglich ihrer Attribute entsprechend definiert ist,
- $\pi_2(mp) = EF_{mp}, EF_{mp} \subseteq EF$, die Effektelemente des Planlets,
- $F := \{f : TE_f \rightarrow \wp(EF), TE_f \subseteq TE\}$ die Menge aller Abbildungen, die bestimmten Topologieelementen eine Menge an Effektelementen zuordnen,
- $\pi_3(mp) = effekte_{mp}, effekte_{mp} \in F$, $effekte_{mp} : TE_{tf_{mp}} \rightarrow \wp(EF_{mp})$, eine Abbildung, die jedem Element $te_i \in TE_{tf_{mp}}$ die Effektelemente zuweist, welche die Effekte, die durch Ausführung von im_{mp} auf der von te_i repräsentierten Komponente bzw. Relation verursacht werden, repräsentieren,
- IM die Menge aller imperativen Managementmodelle,
- $\pi_4(mp) = im_{mp}, im_{mp} \in IM$, das als traditioneller Subprozess realisierte imperative Managementmodell, das die in tf_{mp} beschriebenen Aufgaben ausführt und die Mechanismen aus [Abschnitt 5.2.4](#) implementiert. ■

5.3 Generische Management-Planlets

Die bisher diskutierten Management-Planlets definieren ihre Anwendbarkeit durch vollständig typisierte Anwendungsstrukturen, d. h. jedes Topologieelement des Topologiefragments spezifiziert eindeutig den Typ der repräsentierten Komponente bzw. Relation. Diese *streng typisierten Planlets* sind daher exakt an die im Topologiefragment modellierten Elementkombinationen gekoppelt und können dadurch dedizierte, speziell auf diese Struktur abgestimmte Managementlogik implementieren. Durch dieses Konzept sind komplexe Integrationsszenarien verschiedener Technologien realisierbar, um eine bestimmte Annotation zu implementieren, beispielsweise um proprietäre Management-APIs, komplexe Skripte oder „Legacy“-Managementsoftware zu integrieren.

In vielen Szenarien, typischerweise bei Bereitstellungsaufgaben und einfachen Managementaufgaben wie Daten Im- oder Export, ist jedoch häufig nur die Ausführung einer einzigen, geläufigen Managementtechnologie nötig, um das deklarierte Ziel zu erreichen. Speziell für die Installation von Software bieten zahlreiche Konfigurationsmanagementwerkzeuge eine Vielzahl an erprobten ausführbaren Bausteinen, welche die gesamte Managementaufgabe durch eine de-facto standardisierte Ausführungslogik abdecken können. Um diese auszuführen bedarf es somit keiner individuellen Integrationslogik, die auf einen bestimmten Komponenten- oder Relationentyp zugeschnitten ist, sondern lediglich die generische Ausführung eines bestimmten Artefakts durch eine häufig eingesetzte Technologie. Um den Entwicklungsaufwand für neue Komponententypen, Relationentypen und Managementannotationen zu reduzieren, wird in diesem Abschnitt das Konzept der *generischen Management-Planlets* eingeführt, welche unabhängig von konkreten Typen auf verschiedene Komponenten und Relationen zur Ausführung bestimmter Managementannotationen eingesetzt werden können. Im Vergleich zu streng typisierten Planlets ist diese generische Form nicht vollständig an konkrete Komponenten- oder Relationentypen gebunden, sondern an formal definierte Managementschnittstellen, wie beispielsweise die Lebenszyklusschnittstelle, Eigenschaften und Artefakte. Diese ermöglichen es dem Planlet, eine generische Standardausführung zu imple-

mentieren, welche lediglich von diesen Informationen abhängt und dadurch unabhängig von den eigentlichen Komponenten und Relationen realisiert werden kann. Das Topologiefragment eines generischen Planlets spezifiziert daher, im Vergleich zu streng typisierten Planlets, Vorbedingungen in generischer Art und Weise basierend auf Schnittstellen, Eigenschaften und Artefakten. Konzeptionell entsprechen generische Planlets streng typisierten Planlets, d. h. sie folgen demselben Metamodell. Generische Planlets werden wie folgt definiert:

Definition 54 (Generisches Management-Planlet). Ein generisches Management-Planlet $gmp \in GMP$, $GMP \subset MP$, ist ein Management-Planlet, das für mindestens eine in seinem annotierten Topologiefragment repräsentierte Komponente oder Relation keinen Komponenten- bzw. Relationentyp definiert. Somit erfüllt dessen Topologiefragment tf_{gmp} folgende Aussage:

$$\exists te \in TE_{tf_{gmp}} : typ_{tf_{gmp}}(te) = \perp$$

■

Abbildung 5.6 zeigt ein Beispiel. Das Topologiefragment beschreibt die Bereitstellung einer Komponente auf einer virtuellen Maschine des Typs *Ubuntu12.04*. Dabei ist lediglich die virtuelle Maschine typisiert, der Typ der bereitzustellenden Komponente ist undefiniert, was durch das Symbol $\perp$ modelliert wird. Vorbedingungen an die virtuelle Maschine sind deren Existenz sowie nicht-leere Eigenschaften *IP* und *SSHCredentials*, was durch das Symbol $*$ für den Wert der Eigenschaft ausgedrückt wird. Die untypisierte Komponente muss die Lebenszyklus-Managementschnittstelle *LifecycleManagement* mit der Operation *Install+* bereitstellen, welche durch eine Operationsimplementierung des Typs *BashScript* implementiert wird (vgl. Abschnitt 4.3). Der Effekt der Planlet-Ausführung wird durch die *State*-Eigenschaft an der bereitzustellenden Komponente ausgedrückt, welche deren Instanziierung deklariert (Wert der Eigenschaft wird auf „Running“ gesetzt). Das imperative Managementmodell liest die als Vorbedingung deklarierten Informationen der virtuellen Maschine aus, um das Bash-Skript via SSH auf diese zu kopieren. Anschließend werden die Eingabeparameter der Operation aus dem Anwendungsinstanzmodell gelesen,

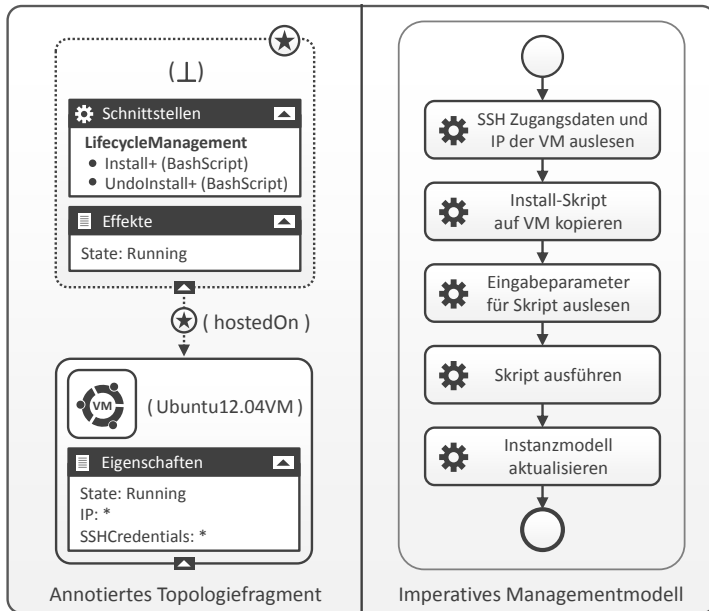


Abbildung 5.6: Generisches Management-Planlet zur Bereitstellung von Komponenten (Ohne Darstellung der Kompensationslogik)

das Skript entsprechend parametrisiert und ausgeführt. Im letzten Schritt werden alle Änderungen, welche durch die Skript-Ausführung an der Anwendung getätigt wurden, in das Instanzmodell übertragen. Der beschriebene Prozess ist dabei lediglich an das Betriebssystem, die Managementschnittstelle und deren Implementierung sowie an Eigenschaften gekoppelt, nicht jedoch an den Typ der bereitzustellenden Komponente. Auf Basis dieses generischen Management-Planlets können somit beliebige Komponenten auf Ubuntu-Betriebssystemen installiert werden, sofern dafür ein Bash-Skript existiert. Um eventuell gescheiterte Installationen durch die in [Abschnitt 5.2.4](#) vorgestellten Mechanismen rückgängig zu machen, definiert das Planlet eine zusätzliche Vorbedingung in Form einer Operation namens *UndoInstall+*, mithilfe dessen Ausführung der Zustand vor Ausführung des Install-Skripts wiederhergestellt werden kann.

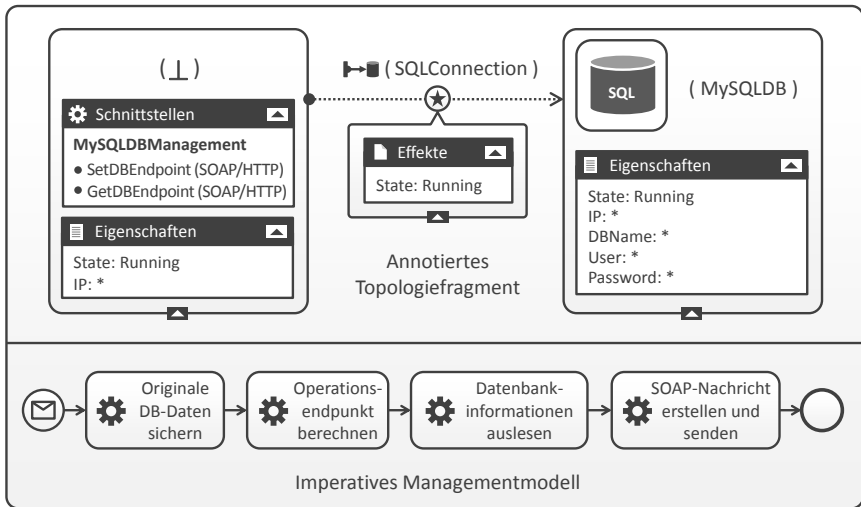


Abbildung 5.7: Generisches Management-Planlet zur Herstellung einer SQL-Verbindung (Ohne Darstellung der Kompensationslogik)

Neben der Bereitstellung können über dieses Prinzip verschiedenste Typen an Managementaufgaben auf generische Weise von individuellen Komponenten- und Relationentypen entkoppelt werden. Die Vorbedingungen, welche zur Ausführung der Annotationen erforderlich sind, müssen dazu klar in Form von Managementschnittstellen, Artefakten und Eigenschaften definiert werden.

Ein weiteres Beispiel ist die Definition einer generischen Datenbankschnittstelle, welche die Semantik der Operation „SetDBEndpoint“, deren Eingabeparameter (*Benutzername*, *Passwort*, *Endpunkt* und *Name* der Zieldatenbank) sowie von Nachrichtenformaten spezifiziert, um die Datenbankzugangsdaten einer Komponente zu konfigurieren. [Abbildung 5.7](#) zeigt ein Planlet, das diese Operation mittels eines vordefinierten SOAP-Nachrichtenformats über HTTP an einer beliebigen Komponente aufrufen kann, um eine SQL-Verbindung zu einer MySQL-Datenbank herzustellen. Das Planlet definiert für diese Komponente, neben der Existenz der Schnittstelle und IP-Adresse, auch Vorbedingungen an die Datenbank in Form der Zugangsdaten, Datenbankadresse sowie dem Daten-

banknamen. Das Planlet ist daher auf beliebige Komponenten und Datenbanken vom Typ „MySQLDB“ anwendbar, sofern diese durch eine bereitzustellende SQL-Verbindung verbunden sind und die Vorbedingungen erfüllen.

Das imperative Managementmodell verwendet diese als Vorbedingungen definierten Informationen, um die SQL-Verbindung herzustellen. Dafür wird zuerst der Endpunkt (d. h. die URL) der SOAP-basierten „SetDBEndpoint“-Operation aus der IP-Adresse der Komponente und dem *Daten*-Attribut der Operationsimplementierung abgeleitet. Dieses Attribut wird in diesem Fall zur Spezifikation des relativen Pfades verwendet, unter welchem die Operation aufrufbar ist (beispielsweise „/MySQLDBManagement“). Anschließend werden die benötigten Datenbankinformationen ausgelesen und mit diesen die SOAP-Nachricht erstellt, welche zur Übermittlung der Informationen an die Operation via HTTP gesendet wird. Um die ursprüngliche Datenbankverbindung im Falle eines unbeheblichen Fehlers wiederherzustellen, ruft das Management-Planlet vor Ausführung der genannten Aktivitäten die „GetDBEndpoint“-Operation an der Komponente auf, um die originalen Endpunktinformationen zu sichern. Dazu können Planlets beispielsweise zusätzliche Informationen im Instanzmodell der Anwendung hinterlegen, um diese im Falle eines Abbruchs abzurufen. Muss die Ausführung rückgängig gemacht werden, ruft das Planlet die „SetDBEndpoint“-Operation mit den gespeicherten Originalinformationen auf und stellt den Ursprungszustand wieder her. Dies wird detailliert in [Abschnitt 5.5](#) im Rahmen der Funktionsweise und Implementierung von Planlets erläutert.

Die beiden Beispiele zeigen, dass generische Planlets sowohl zur Ausführung von Managementaufgaben auf Komponenten als auch auf Relationen entwickelt werden können. Durch die Möglichkeit, die Semantik von Schnittstellen, Eigenschaften und Artefakten unabhängig von spezifischen Komponenten- und Relationentypen zu definieren, ist dieser Ansatz flexibel erweiterbar und kann zur effizienten Entwicklung wiederverwendbarer Managementbausteine herangezogen werden. Dadurch wird die Kopplung an individuelle Komponenten- und Relationentypen durch generische Vorbedingungen gelöst.

5.4 Orchestrierung von Management-Planlets

In diesem Abschnitt wird ein Verfahren vorgestellt, das die vollautomatisierte Transformation deklarativer DMMN-Modelle in ausführbare imperative Managementmodelle mittels Orchestrierung geeigneter Management-Planlets ermöglicht. Das vorgestellte Transformationsverfahren dient daher zur Automatisierung von Schritt 4 der PALMA-Methode („Transformation in imperatives Managementmodell“, vgl. [Abschnitt 3.3.5](#)). Zudem wird die Komponente des *Plangenerators* eingeführt, welche dieses Verfahren implementiert.

Im nachfolgenden Unterabschnitt [5.4.1](#) wird zunächst eine Methode vorgestellt, die einen Überblick über die Verfahrensschritte gibt. Zur Umsetzung dieser Methode werden in [Abschnitt 5.4.2](#) zunächst Kompatibilitätsrelationen formalisiert, welche die Eignung eines Management-Planlets zur Ausführung einer oder mehrerer Managementannotationen innerhalb eines DMMN-Modells evaluieren. [Abschnitt 5.4.3](#) stellt den Gesamtalgorithmus zur Transformation deklarativer DMMN-Modelle in imperative Managementmodelle vor, der aus weiteren Subroutinen besteht, die in den folgenden Kapiteln vorgestellt werden. Die Anwendung der definierten Kompatibilitätsregeln wird in [Abschnitt 5.4.4](#) durch einen Algorithmus automatisiert, der die Kompatibilität eines Planlets bezüglich eines DMMN-Modells evaluiert und Zuordnungen erstellt, welche die Topologie- und Annotationenelemente des Topologiefragments auf die entsprechenden Elemente des DMMN-Modells abbilden. Dieser Algorithmus wird bei der Orchestrierung von Management-Planlets zur Berechnung einer Kandidatenmenge an Planlets eingesetzt, welche die im DMMN-Modell deklarierten Annotationen potentiell ausführen können. In [Abschnitt 5.4.7](#) wird ein weiterer Algorithmus vorgestellt, der die (i) Auswahl und (ii) Herstellung einer partiellen Ordnung (engl. „*partial order*“) von Management-Planlets automatisiert durchführt. In [Abschnitt 5.5.1](#) wird die technologieunabhängige Transformation dieser Ordnung in ein ausführbares imperatives Managementmodell beschrieben, die in [Abschnitt 5.5.2](#) auf die Workflow-Sprache BPEL angewendet wird. Alle in diesem Kapitel vorgestellten Algorithmen werden im Rahmen der prototypischen Realisierung des PALMA-Frameworks in [Kapitel 8](#) implementiert.

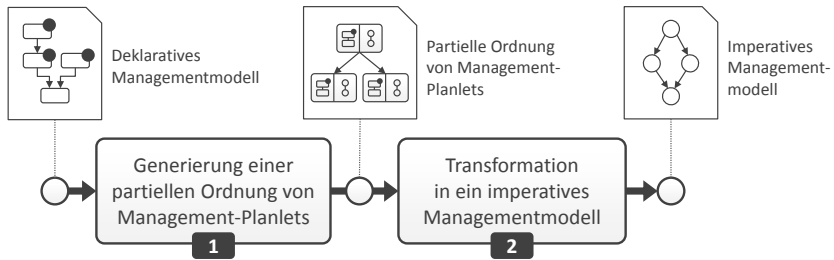


Abbildung 5.8: Transformationsmethode

5.4.1 Transformationsmethode

Die Transformation von DMMN-Modellen in imperative Managementmodelle besteht aus zwei Schritten und wird in [Abbildung 5.8](#) als Methode dargestellt:

Im ersten Schritt wird das deklarative Managementmodell analysiert und für die darin deklarierten Annotationen Management-Planlets ausgewählt, welche diese ausführen können, sowie deren Ausführungsreihenfolge spezifiziert. Die Auswahl der Planlets basiert dabei auf der Kompatibilität deren Vorbedingungen und Effekte zueinander sowie zum DMMN-Modell, wofür im folgenden Abschnitt formale Kompatibilitätsrelationen definiert werden. Auf Basis dieser Auswahl wird die Ausführungsreihenfolge der Planlets durch einen Algorithmus zur Berechnung der partiellen Ordnung abgeleitet (siehe [Abschnitt 5.4.7](#)).

Im zweiten Schritt werden die partiell geordneten Planlets in ein imperatives Managementmodell transformiert, welches alle im DMMN-Modell spezifizierten Managementannotationen auf der realen Anwendung ausführt. Imperative Managementmodelle werden in der vorliegenden Arbeit in Form von imperativen, graphbasierten Workflow-Modellen realisiert, die gemäß [Abschnitt 2.3.4.3](#) als Managementpläne bezeichnet werden. Dabei ist eine Kernanforderung die Atomarität der generierten Managementpläne. Diese werden entweder vollständig erfolgreich ausgeführt oder überführen die Anwendung im Falle eines Fehlers wieder in ihren Ursprungszustand. Zur praktischen Validierung dieses Verfahrens wird in [Abschnitt 5.5.2](#) die Transformation partieller Ordnungen BPEL-basierter Planlets in ausführbare BPEL-Managementpläne vorgestellt.

5.4.2 Kompatibilitätsdefinitionen

Der erste Schritt der vorgestellten Transformationsmethode stellt eine partielle Ordnung von Management-Planlets her, welche alle im DMMN-Modell deklarierten Managementannotationen ausführen. Um ein Management-Planlet zur Ausführung einer oder mehrerer Managementannotationen in einem DMMN-Modell einzusetzen, müssen die Vorbedingungen des Topologiefragments des Planlets erfüllt werden und die darin spezifizierten Managementannotationen kompatibel zu den Managementannotationen des DMMN-Modells sein. In diesem Kapitel wird dazu die Kompatibilität von Instanzen der DMMN-Klassen definiert, mithilfe derer sich feststellen lässt, ob ein Management-Planlet zur Ausführung einer oder mehrerer Managementannotationen eines bestimmten DMMN-Modells eingesetzt werden kann. Die eingeführten Definitionen formalisieren dadurch die bisher ausschließlich informell diskutierten Vorbedingungen von Management-Planlets, präzisieren diese und werden durch einen Algorithmus zur Bestimmung der Kompatibilität automatisiert angewendet ([Abschnitt 5.4.5](#)). Im Folgenden werden die verschiedenen Arten von Vorbedingungen klassifiziert und deren Kompatibilität informell definiert, welche anschließend als Grundlage zur formalen Kompatibilitätsdefinition dient.

Generell werden drei disjunkte Klassen an Vorbedingungen eines Management-Planlets unterschieden: (i) *Strukturelle Vorbedingungen*, (ii) *aufgabenbezogene Vorbedingungen* und (iii) *zustandsbezogene Vorbedingungen*. Die erste Klasse definiert Vorbedingungen bezüglich der Topologie, auf die ein Planlet angewendet werden kann. Diese umfassen die Struktur des DMMN-Modells, die Typen der enthaltenen Topologielemente, deren Schnittstellen, Operationen sowie Artefakte. Die *strukturelle Kompatibilität* wird folgendermaßen definiert:

Definition 55 (Strukturelle Kompatibilität - informell). Ein Management-Planlet ist bezüglich der Ausführung einer oder mehrerer Managementannotationen *strukturell* zu einem DMMN-Modell kompatibel, wenn die im Topologiefragment modellierten Vorbedingungen in Form der Topologiestruktur, deren Typ-, Schnittstellen-, Operationen- und Artefaktelementen erfüllt sind. ■

Die zweite Klasse definiert Vorbedingungen bezüglich deklarierter Managementannotationen, deren Konfigurationen und Ausführungsreihenfolgen. Die *aufgabenbezogene Kompatibilität* wird wie folgt informell definiert:

Definition 56 (Aufgabenbezogene Kompatibilität - informell). Ein Planlet ist bezüglich der Ausführung von Managementannotationen *aufgabenbezogen* zu einem DMMN-Modell kompatibel, wenn die im Topologiefragment modellierten Annotationen sowie deren Konfigurationen kompatibel zu Annotationenelementen des DMMN-Modells sind und alle diesbezüglich im DMMN-Modell deklarierten Reihenfolgespezifikationen vom Planlet beachtet werden. ■

Beide Klassen definieren folglich Vorbedingungen, welche nicht durch die Ausführung von Management-Planlets beeinflusst werden: Ein Planlet kann weder die Struktur eines DMMN-Modells verändern noch die darin repräsentierten Typen, Managementschnittstellen, Annotationen, Konfigurationen, Reihenfolgespezifikationen, Operationen oder Artefakte modifizieren, entfernen oder neue Elemente hinzufügen. Daher werden diese Informationen ausschließlich vom DMMN-Modell spezifiziert und sind somit *statischer* Natur, da sie während der Managementdurchführung nicht durch Management-Planlets verändert werden können. Auf Basis dieser *statischen Vorbedingungen* kann somit evaluiert werden, ob ein Planlet grundsätzlich zur Ausführung bestimmter Managementannotationen eines DMMN-Modells eingesetzt werden kann.

Die dritte Klasse definiert Anforderungen an den Zustand der Anwendung, d. h. an die Eigenschaften der Komponenten und Relationen, welche ein Planlet durch sein Topologiefragment als *zustandsbezogene Vorbedingungen* für dessen Ausführung definieren kann. Da Planlets Eigenschaften zur Übertragung von Laufzeitinformationen verwenden (vgl. [Abschnitt 5.2.3](#)), unterliegen diese typischerweise einer hohen Änderungsrate und sind daher *dynamischer* Natur.

Definition 57 (Zustandsbezogene Kompatibilität - informell). Ein Management-Planlet ist bezüglich der Ausführung einer Managementannotation *zustandsbezogen* zu einem DMMN-Modell kompatibel, wenn die im Topologiefragment als Eigenschaftenelemente spezifizierten Vorbedingungen erfüllt sind. ■

Während statische Vorbedingungen bereits vor der Generierung der partiellen Ordnung zur Berechnung einer Kandidatenmenge grundsätzlich anwendbarer Planlets evaluiert werden können, beeinflussen zustandsbezogene Vorbedingungen die letztendliche Auswahl der tatsächlich verwendeten Planlets erst während des Generierungsprozesses, da diese von zuvor auszuführenden Planlets und den resultierenden Anwendungszuständen abhängig sind. Stehen beispielsweise für eine Annotation $ma \in MA_d$ eines DMMN-Modells $d \in D$ zwei Planlets $mp_1, mp_2 \in MP$ zur Verfügung, deren statische Vorbedingungen identisch sind und sich ausschließlich durch den Wert einer einzigen geforderten Eigenschaft unterscheiden, kann ein drittes Planlet, welches vor diesen Planlets ausgeführt wird und die differenzierende Eigenschaft überschreibt, die Auswahl beeinflussen. Ob mp_1 oder mp_2 zur Ausführung der von ma repräsentierten Annotation verwendet werden kann ist somit nicht vorab entscheidbar. Daher können statische Vorbedingungen unabhängig von anderen Planlets evaluiert werden, die Entscheidung, welches Planlet letztendlich in welcher Reihenfolge ausgeführt wird, kann jedoch erst bei der Generierung der partiellen Ordnung durch Evaluation der zustandsbezogenen Kompatibilität geschehen.

Um ein Management-Planlet zur Ausführung einer Annotation eines DMMN-Modells einzusetzen, muss dieses daher sowohl *statisch*, d. h. strukturell und aufgabenbezogen, als auch *dynamisch*, d. h. zustandsbezogen, kompatibel zum DMMN-Modell bzw. zum Anwendungszustand des Ausführungszeitpunktes sein. Auf Basis der informellen Definitionen werden in den folgenden Abschnitten zunächst formale Kompatibilitätsrelationen bezüglich der statischen Kompatibilität folgender Klassen eingeführt: Artefakte, Managementoperationen, Managementschnittstellen, Annotationen und Topologieelemente. Alle eingeführten Relationen sind dabei zweistellig sowie homogen und werden durch das $\approx_x$ Symbol in der Infix-Notation beschrieben. Das Subskript „x“ definiert die DMMN-Klasse, d. h. die Quell- und Zielmenge der Relation. Die Definitionen dienen in [Abschnitt 5.4.4](#) zur Berechnung aller zu einem DMMN-Modell statisch kompatiblen Planlets. Die dynamische Kompatibilität auf Basis von Eigenschaften wird im Rahmen der Plangenerierung in [Abschnitt 5.4.7.1](#) definiert.

5.4.2.1 Kompatibilität von Artefakten

Anwendungsartefakte unterliegen derselben Kompatibilitätsdefinition wie Operationsimplementierungen. Daher wird die Kompatibilität auf Basis deren abstrakter DMMN-Superklasse „Artefakt“ definiert. Das „Name“-Attribut eines Artefakts wird zur Definition der semantischen Bedeutung verwendet. Dies spielt vor allem bei Anwendungsartefakten eine Rolle, da beispielsweise durch den Namen „Installable“ klar spezifiziert werden kann, dass es bei einem Anwendungsartefakt um eine Installationsdatei handelt, dessen Format durch das „Datentyp“-Attribut beschrieben ist. Dadurch kann ein Management-Planlet die Existenz eines bestimmten Anwendungsartefakts als Vorbedingung in Form der Attribute „Name“ und „Datentyp“ eindeutig definieren. Wird die Bereitstellungslogik einer Komponente oder Relation jedoch vollständig durch eine einzelne Operationsimplementierung durchgeführt, spielt der Datentyp involvierter Anwendungsartefakte aus Sicht des Management-Planlets möglicherweise keine Rolle. Beispielsweise, wenn Anwendungsartefakte direkt von einer Operationsimplementierung verarbeitet werden und ein generisches Planlet diese Operationsimplementierung lediglich aufruft. Daher kann in einigen Fällen auf die Spezifikation des Typs bei Artefakten als Vorbedingung verzichtet werden.

In ähnlicher Form spielt das „Name“-Attribut bei einzelnen Operationsimplementierungen einer Operation eine untergeordnete Rolle, da die funktionale Semantik der Operation durch den Typ der Operation definiert wird und dafür oftmals genau eine Implementierung zur Verfügung steht. In diesem Fall hat das „Name“-Attribut der Operationsimplementierung lediglich informative Bedeutung, beeinflusst jedoch nicht die operationale Semantik des Modells oder deren schlussendliche Ausführung. Wird eine Operation jedoch durch mehrere Operationsimplementierungen realisiert, kann das „Name“-Attribut zur Unterscheidung derer verwendet werden. Da die folgenden Definitionen auch bei der Automatisierung von Managementmustern in [Kapitel 6](#) Verwendung finden, werden diese unabhängig von Planlets ausschließlich auf Basis von DMMN-Modellen definiert. Zur klaren Unterscheidung stellt $d_1 \in D$ in diesem Kapitel durchgehend das Topologiefragment des Planlets dar, $d_2 \in D$ das DMMN-Modell, für welches dessen Anwendbarkeit evaluiert werden soll.

Gemäß diesen Regeln wird die Kompatibilitätsrelation $\tilde{\approx}_a$ wie folgt definiert:

Definition 58 (Kompatibilität von Artefaktelementen). Seien $a_1 \in A_{d_1}$ und $a_2 \in A_{d_2}$ Artefaktelemente zweier DMMN-Modelle $d_1, d_2 \in D$, dann wird die Kompatibilitätsrelation $\tilde{\approx}_a \subseteq A \times A$ folgendermaßen definiert:

$$a_1 \tilde{\approx}_a a_2 :\Leftrightarrow (\pi_2(a_1) = \pi_2(a_2) \vee \pi_2(a_1) = \perp) \wedge \\ (\pi_4(a_1) = \pi_4(a_2) \vee \pi_4(a_1) = \perp)$$

■

5.4.2.2 Kompatibilität von Managementoperationen

Um die Kompatibilität der Managementoperationen zweier Managementschnittstellen zu evaluieren, müssen gewisse Eigenschaften erfüllt sein, welche vom Management-Planlet als Vorbedingungen bezüglich der Operation definiert sind. Die erste Bedingung ist die Existenz einer Operation gleichen Typs auf Seite des DMMN-Modells, welche durch kompatible Artefakte implementiert ist. Um eine Managementoperation auszuführen, müssen deren Eingabeparameter vom Aufrufenden spezifiziert werden. Das Topologiefragment des Management-Planlets definiert hierzu dem Planlet bekannte Eingabeparameter bezüglich der Operation, welche es bei Ausführung der Operation setzen kann. Für alle Parameter, die bezüglich des Namens und des Typs übereinstimmen, ist die Kompatibilität gewährleistet. Definiert die Operation des DMMN-Modells weitere Eingabeparameter, welche das Planlet nicht in seinen Vorbedingungen spezifiziert, kann die Operation nur ausgeführt werden, falls die unbekannten Parameter Eigenschaftsreferenzen spezifizieren. Diese können von jedem Planlet aus dem Anwendungsinstanzmodell abrufen werden, um die Parametrisierung des Operationsaufrufes entsprechend durchzuführen. In beiden Fällen ist das Planlet für die Wertzuweisung des Parameters beim Aufruf der Operation verantwortlich. Diese Regeln werden in folgender Definition formalisiert:

Definition 59 (Kompatibilität von Operationenelementen). Seien $o_1 \in O_{d_1}$ und $o_2 \in O_{d_2}$ Operationenelemente zweier DMMN-Modelle $d_1, d_2 \in D$, dann wird die Kompatibilitätsrelation $\tilde{\approx}_o \subseteq O \times O$ folgendermaßen definiert:

$$\begin{aligned}
o_1 \tilde{\approx}_o o_2 : & \Leftrightarrow \pi_2(o_1) = \pi_2(o_2) \wedge \\
& \forall a_i \in \text{eigeneArtefakte}_{d_1}(o_1) \exists a_j \in \text{eigeneArtefakte}_{d_2}(o_2): a_i \tilde{\approx}_a a_j \wedge \\
& \left(\forall \text{param}_n \in \pi_3(o_2): \left(\pi_2(\text{param}_n) \neq \perp \right) \vee \right. \\
& \left. \left(\exists \text{param}_k \in \pi_3(o_1): \pi_1(\text{param}_n) = \pi_1(\text{param}_k) \wedge \right. \right. \\
& \quad \left. \left. \pi_3(\text{param}_n) = \pi_3(\text{param}_k) \right) \right)
\end{aligned}$$

■

5.4.2.3 Kompatibilität von Managementschnittstellen

Da Managementoperationen immer Teil einer Managementschnittstelle sind, erfordert die Kompatibilität zweier Managementschnittstellen auch die Überprüfung der Kompatibilität deren Operationen. Auf Basis der Kompatibilitätsrelation $\tilde{\approx}_o$ kann die Kompatibilität zwischen Managementschnittstellen wie folgt definiert werden: Spezifiziert ein Management-Planlet eine Managementschnittstelle als Vorbedingung, muss das zugehörige Topologieelement des DMMN-Modells, für welches die Anforderung gilt, eine Managementschnittstelle mit identischem „Typ“ aufweisen. Zudem muss für jede Managementoperation der Schnittstelle des Management-Planlets eine kompatible Operation innerhalb der Managementschnittstelle des DMMN-Modells vorhanden sein. Die Kompatibilität zwischen Managementschnittstellen wird auf Basis dieser informellen Regeln folgendermaßen formal definiert:

Definition 60 (Kompatibilität von Managementschnittstellen). Seien $i_1 \in I_{d_1}$ und $i_2 \in I_{d_2}$ Managementschnittstellenelemente zweier DMMN-Modelle $d_1 \in D$ und $d_2 \in D$, dann wird die Kompatibilitätsrelation $\tilde{\approx}_i \subseteq I \times I$ folgendermaßen formal definiert:

$$i_1 \tilde{\approx}_i i_2 :\Leftrightarrow \pi_2(i_1) = \pi_2(i_2) \wedge \\ \forall o_i \in \text{operationen}_{d_1}(i_1) \exists o_j \in \text{operationen}_{d_2}(i_2) : o_i \tilde{\approx}_o o_j$$

■

5.4.2.4 Kompatibilität von Managementannotationen

Zwei Managementannotationen sind kompatibel, wenn (i) deren „Typ“-Attribute übereinstimmen, z. B. dass es sich jeweils um eine Bereitstellungsannotation handelt, und (ii) falls die spezifizierte Konfiguration kompatibel zur Konfiguration der zu vergleichenden Annotation ist, was durch Äquivalenz der Konfigurationstypen gewährleistet wird. Die Konfigurationsparameter werden dabei nicht beachtet, da diese inhärent durch den Konfigurationstyp definiert sind. Diese informellen Regeln werden folgendermaßen formalisiert:

Definition 61 (Kompatibilität von Annotationenelementen). Seien $ma_1 \in MA_{d_1}$ und $ma_2 \in MA_{d_2}$ Annotationenelemente zweier DMMN-Modelle $d_1, d_2 \in D$, dann wird die Kompatibilitätsrelation $\tilde{\approx}_{ma} \subseteq MA \times MA$ folgendermaßen definiert:

$$ma_1 \tilde{\approx}_{ma} ma_2 :\Leftrightarrow \pi_2(ma_1) = \pi_2(ma_2) \wedge \\ \left((konfiguration_{d_2}(ma_2) \neq \perp) \Rightarrow (konfiguration_{d_1}(ma_1) \neq \perp \wedge \right. \\ \left. (\pi_2(konfiguration_{d_1}(ma_1)) = \pi_2(konfiguration_{d_2}(ma_2)) \right. \\ \left. \vee \pi_2(konfiguration_{d_1}(ma_1)) = *) \right) \right)$$

■

5.4.2.5 Kompatibilität von Topologieelementen

Basierend auf den vorangegangenen Definitionen wird die Kompatibilität von Topologieelementen zwischen DMMN-Modellen folgendermaßen informell definiert: Für ein Topologieelement, das im Topologiefragment des Planlets enthalten ist, definiert das Fragment Anforderungen in Form von (i) Schnittstellen, (ii) Artefakten sowie dem (iii) Typ des Topologieelements, welche zur Ausführung des Management-Planlets erfüllt sein müssen. Ein Topologieelement, das im auszuführenden DMMN-Modell enthalten ist, muss für jede dieser Vorbedingungen jeweils einen kompatiblen Gegenpart spezifizieren, um kompatibel zum Element des Fragments zu sein. Für das Topologiefragment des Planlets werden nur die eigenen Managementschnittstellen und Artefakte der Topologieelemente einbezogen, um eine präzise Modellierung der Vorbedingungen zu ermöglichen und geerbte Schnittstellen bzw. Artefakte auszuschließen.

Managementannotationen werden in dieser Kompatibilitätsrelation nicht berücksichtigt, da diese aufgrund möglicher Reihenfolgespezifikationen, welche Topologiefragment-übergreifend modelliert sein können, im Gesamtkontext des DMMN-Modells evaluiert werden müssen (siehe [Abschnitt 5.4.2.6](#)). Gleichmaßen werden auch Zustandsinformationen in der Form von Eigenschaftenelementen nicht in diese Kompatibilitätsrelation einbezogen, da diese nicht nur vom DMMN-Modell abhängig sind, sondern auch von zuvor ausgeführten Planlets. Die im Folgenden definierte Kompatibilitätsrelation $\tilde{\approx}_{te}$ berücksichtigt somit ausschließlich die strukturelle Kompatibilität zweier Topologieelemente.

Definition 62 (Kompatibilität von Topologieelementen). Seien $te_1 \in TE_{d_1}$ und $te_2 \in TE_{d_2}$ Topologieelemente zweier DMMN-Modelle $d_1, d_2 \in D$, dann wird die Kompatibilitätsrelation $\tilde{\approx}_{te} \subseteq TE \times TE$ folgendermaßen formal definiert:

$$\begin{aligned} te_1 \tilde{\approx}_{te} te_2 &:= \left(\pi_2(\text{typ}_{d_1}(te_1)) \in \pi_2(\text{supertypen}(te_2)) \vee \text{typ}_{d_1}(te_1) = \perp \right) \wedge \\ &\quad \left(\forall i_i \in \text{eigeneSchnittstellen}_{d_1}(te_1) \exists i_j \in \text{schnittstellen}(te_2) : i_i \tilde{\approx}_{i_j} i_j \right) \wedge \\ &\quad \left(\forall a_n \in \text{eigeneArtefakte}_{d_1}(te_1) \exists a_m \in \text{artefakte}(te_2) : a_n \tilde{\approx}_a a_m \right) \end{aligned}$$

■

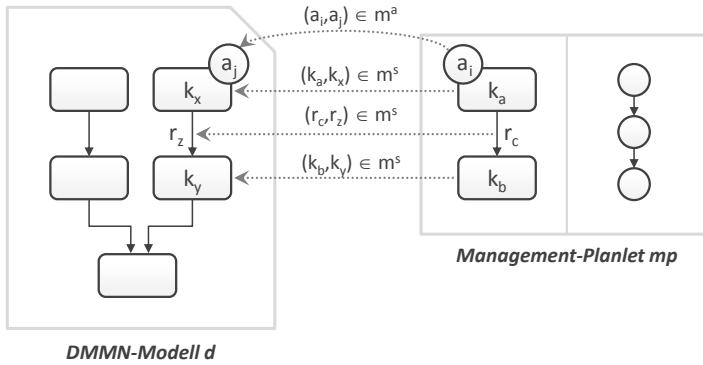


Abbildung 5.9: Strukturelle und aufgabenbezogene Elementzuordnungen zwischen DMMN-Modellen zur Evaluation deren Kompatibilität

5.4.2.6 Statische Kompatibilität von Planlets und DMMN-Modellen

Die statische Kompatibilität zwischen Management-Planlets und DMMN-Modellen wird auf Basis der Existenz kompatibler Zuordnungen der Topologie- und Annotationenelemente des Topologiefragments eines Planlets auf die entsprechenden Elemente des DMMN-Modells definiert. Dazu werden wie in [Abbildung 5.9](#) dargestellt (i) strukturelle und (ii) aufgabenbezogene Zuordnungen eingeführt. Da die folgenden Definitionen ebenso bei der Automatisierung von Managementmustern in [Kapitel 6](#) Verwendung finden, werden diese unabhängig von Planlets auf Basis von DMMN-Modellen definiert. Zur klaren Unterscheidung stellt d_1 in diesem Abschnitt durchgehend das Topologiefragment eines Management-Planlets dar, d_2 ein DMMN-Modell, für welches die statische Kompatibilität des Planlets evaluiert werden soll, d. h. ob es grundsätzlich auf d_2 anwendbar ist.

Definition 63 (Strukturelle Zuordnung). Sei TE die Menge aller Topologieelemente, dann ist eine *strukturelle Zuordnung* $m^s \in \mathfrak{M}^s, m^s \subseteq TE \times TE$, eine links- und rechtseindeutige Relation, deren Elemente $(te_1, te_2) \in m^s$ jeweils ein Topologieelement $te_1 \in TE$ auf ein Topologieelement $te_2 \in TE$ abbilden:

$$\mathfrak{M}^s := \wp(TE \times TE)$$

■

Strukturelle Zuordnungen werden zur Abbildung der Topologieelemente eines DMMN-Modells $d_1 \in D$ auf die Topologieelemente eines DMMN-Modells $d_2 \in D$ verwendet. Auf Basis dieser strukturellen Zuordnungen wird die *strukturelle Kompatibilität* zweier DMMN-Modelle wie folgt formal definiert:

Definition 64 (Strukturell kompatible Zuordnung). Seien $d_1, d_2 \in D$ DMMN-Modelle, dann ist d_1 bezüglich einer strukturellen Zuordnung $m^s \in \mathfrak{M}^s$ genau dann strukturell zu d_2 kompatibel, wenn diese bezüglich der Relation $kompatibel^s \subseteq D \times D \times \mathfrak{M}^s$ wie folgt in Beziehung zueinander stehen:

$$(d_1, d_2, m^s) \in kompatibel^s :\Leftrightarrow \left(m^s \subseteq TE_{d_1} \times TE_{d_2} \right) \wedge \\ \left(\forall k_i \in K_{d_1} \exists! k_j \in K_{d_2} : (k_i, k_j) \in m^s \wedge k_i \approx_{te} k_j \right) \wedge \\ \left(\forall r_p \in R_{d_1} \exists! r_q \in R_{d_2} : (r_p, r_q) \in m^s \wedge r_p \approx_{te} r_q \wedge \right. \\ \left. \left(\pi_3(r_p), \pi_3(r_q) \right) \in m^s \wedge \left(\pi_4(r_p), \pi_4(r_q) \right) \in m^s \right)$$

Die strukturelle Zuordnung m^s stellt dabei einen Subgraphisomorphismus dar, welcher die Knotenfärbung und Kantengewichtung der Multigraphen d_1 und d_2 syntaktisch auf Basis der Kompatibilitätsrelation $\approx_{te}$ berücksichtigt. ■

Die aufgabenbezogene Kompatibilität zwischen einem Management-Planlet und einem DMMN-Modell evaluiert, ob das Planlet zur Ausführung einer oder mehrerer Managementannotationen des DMMN-Modells verwendet werden kann (vgl. Definition 56). Um Managementannotationen zweier DMMN-Modelle aufeinander abzubilden, werden aufgabenbezogene Zuordnungen eingeführt:

Definition 65 (Aufgabenbezogene Zuordnung). Sei MA die Menge aller Managementannotationenelemente aller DMMN-Modelle, dann ist eine *aufgabenbezogene Zuordnung* $m^{ab} \in \mathfrak{M}^{ab}$, $m^{ab} \subseteq MA \times MA$, eine zweistellige, links- und rechtseindeutige Relation, deren Elemente $(ma_1, ma_2) \in m^{ab}$ jeweils ein Annotationenelement $ma_1 \in MA$ auf ein Annotationenelement $ma_2 \in MA$ abbilden:

$$\mathfrak{M}^{ab} := \wp(MA \times MA)$$

■

Ob ein Planlet zu einem DMMN-Modell aufgabenbezogen kompatibel ist, hängt neben den Annotationen auch von der Struktur des Topologiefragments und des DMMN-Modells sowie von Reihenfolgespezifikationen im DMMN-Modell ab (vgl. [Abschnitt 4.2.4.2](#)). Ist ein Planlet strukturell und aufgabenbezogen zu einem DMMN-Modell kompatibel, existiert eine *statisch kompatible Zuordnung*:

Definition 66 (Statisch kompatible Zuordnung). Seien $d_1, d_2 \in D$ DMMN-Modelle, dann ist d_1 bezüglich einer strukturellen Zuordnung $m^s \in \mathfrak{M}^s$ und einer aufgabenbezogenen Zuordnung $m^{ab} \in \mathfrak{M}^{ab}$ genau dann statisch zu d_2 kompatibel, wenn die Modelle sowie deren Zuordnungen bezüglich der Relation $kompatibel^\circ \subseteq D \times D \times \mathfrak{M}^s \times \mathfrak{M}^{ab}$ wie folgt in Beziehung zueinander stehen:

$$\begin{aligned}
(d_1, d_2, m^s, m^{ab}) \in kompatibel^\circ &: \Leftrightarrow (d_1, d_2, m^s) \in kompatibel^s \wedge \\
&(\forall ma_i \in MA_{d_1} \exists! ma_j \in MA_{d_2} : (ma_i, ma_j) \in m^{ab} \wedge ma_i \approx_{ma} ma_j) \wedge \\
&(\forall (ma_i, ma_j) \in m^{ab} \exists (te_p, te_q) \in m^s : \\
&\quad ma_i \in annotationen_{d_1}(te_p) \wedge ma_j \in annotationen_{d_2}(te_q)) \wedge \\
&(\forall (ma_1, ma_{1*}), (ma_2, ma_{2*}) \in m^{ab} : (ma_{1*}, ma_{2*}, z) \in P_{d_2} \implies \\
&\quad (z = O \wedge (ma_1, ma_2, O) \in P_{d_1}) \vee \\
&\quad (z = C \wedge ((ma_1, ma_2, C) \in P_{d_1} \vee (ma_2, ma_1, C) \in P_{d_1})) \vee \\
&\quad (z = * \vee (ma_1, ma_2, *) \in P_{d_1} \vee (ma_2, ma_1, *) \in P_{d_1}))
\end{aligned}$$

Sei das Quadrupel $(d_1, d_2, m^s, m^{ab}) \in kompatibel^\circ$, dann wird das Paar (m^s, m^{ab}) als *statisch kompatible Zuordnung* bezeichnet. ■

Die statische Kompatibilität zweier DMMN-Modelle wird daher mit der Existenz mindestens einer statisch kompatiblen Zuordnung gleichgesetzt:

Definition 67 (Statische Kompatibilität von DMMN-Modellen). Seien $d_1, d_2 \in D$ DMMN-Modelle, dann ist d_1 statisch kompatibel zu d_2 , wenn diese bezüglich der Kompatibilitätsrelation $\approx_\circ \subseteq D \times D$ wie folgt in Beziehung stehen:

$$d_1 \approx_\circ d_2 : \Leftrightarrow \exists m^s \in \mathfrak{M}^s \exists m^{ab} \in \mathfrak{M}^{ab} : kompatibel^\circ(d_1, d_2, m^s, m^{ab}) \quad \blacksquare$$

Die Menge aller statisch kompatiblen Zuordnungen zwischen zwei DMMN-Modellen wird wie folgt definiert:

Definition 68 (Menge aller statisch kompatiblen Zuordnungen). Sei $d_1 \in D$ und $d_2 \in D$ jeweils ein DMMN-Modell, dann enthält die Menge $\mathfrak{M}_{d_1, d_2}^{\approx_\circ}$ alle statisch kompatiblen Zuordnungen der Elemente in d_1 auf die Elemente in d_2 :

$$\mathfrak{M}_{d_1, d_2}^{\approx_\circ} := \{(m^s, m^{ab}) \in \mathfrak{M}^s \times \mathfrak{M}^{ab} \mid (d_1, d_2, m^s, m^{ab}) \in \textit{kompatibel}^\circ\} \quad \blacksquare$$

Die statische Kompatibilität wird dazu eingesetzt, die grundsätzliche Anwendbarkeit eines Management-Planlets auf ein bestimmtes DMMN-Modell zu evaluieren, d. h. ob die Managementaufgaben und die Struktur, auf welcher das Planlet die Aufgaben ausführt, kompatibel zu den modellierten Aufgaben und der Struktur des DMMN-Modells sind. Dabei werden als Eigenschaften modellierte Vorbedingungen von Planlets nicht berücksichtigt: Die zustandsbezogene Kompatibilität wird erst bei der Orchestrierung von Management-Planlets während der eigentlichen Plangenerierung miteinbezogen, da diese, wie in [Abschnitt 5.4.2](#) beschrieben, von anderen Planlets *dynamisch* beeinflusst wird. Ein Management-Planlet $mp \in MP$ ist daher genau dann statisch zu einem DMMN-Modell $d \in D$ kompatibel, wenn mindestens eine statisch kompatible Zuordnung der Topologie- und Annotationenelemente des Topologiefragments des Planlets auf entsprechende Elemente des DMMN-Modells existiert. Die Menge aller auf ein DMMN-Modell $d \in D$ anwendbaren Management-Planlets inklusive einer jeweils statisch kompatiblen Zuordnung wird wie folgt definiert:

Definition 69 (Statisch kompatible Management-Planlets). Sei $d \in D$ ein DMMN-Modell, dann enthält die Menge $MP_d^{\approx_\circ}$ alle Paare bestehend aus einem Management-Planlet und einer statisch kompatiblen Zuordnung, mithilfe derer das Topologiefragment tf_{mp} statisch kompatibel auf d abgebildet werden kann.

$$MP_d^{\approx_\circ} := \{(mp, (m^s, m^{ab})) \in MP \times (\mathfrak{M}^s \times \mathfrak{M}^{ab}) \mid (m^s, m^{ab}) \in \mathfrak{M}_{tf_{mp}, d}^{\approx_\circ}\} \quad \blacksquare$$

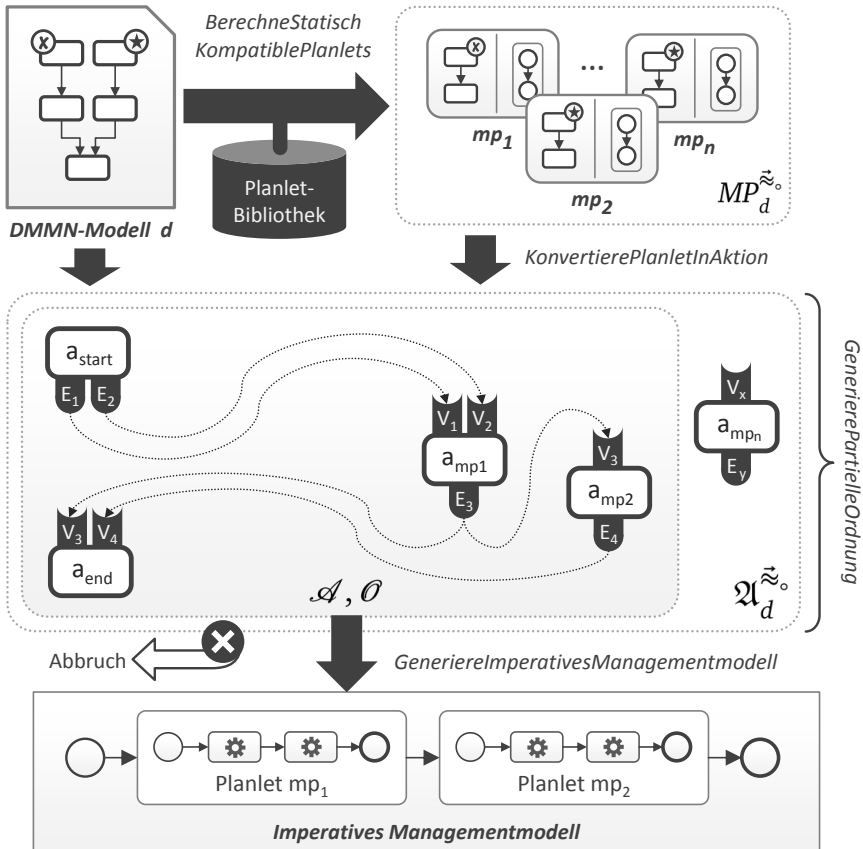


Abbildung 5.10: Illustration des Algorithmus zur Transformation deklarativer DMMN-Modelle in imperative Managementmodelle

5.4.3 Algorithmus zur Plangenerierung

Dieser Abschnitt stellt einen Algorithmus vor, der deklarative DMMN-Modelle in ausführbare imperative Managementmodelle transformiert. Eine grafische Illustration des Algorithmus wird in [Abbildung 5.10](#) dargestellt, [Algorithmus 5.1](#) beschreibt die einzelnen Schritte in Pseudo-Code. Alle darin verwendeten Subroutinen werden in den nachfolgenden Abschnitten im Detail erklärt.

Algorithmus 5.1 TransformationInImperativesManagementmodell ($d \in D$)

```
1:  $\mathcal{A}_d^{\tilde{\circ}} := \emptyset$ 
2:  $MP_d^{\tilde{\circ}} := \text{BerechneStatischKompatiblePlanlets}(d)$ 
3: // Konvertiere alle strukturell und aufgabenbezogen kompatiblen Planlets in Aktionen
4: for all  $((mp, (m^s, m^{ab})) \in MP_d^{\tilde{\circ}})$  do
5:    $\mathcal{A}_d^{\tilde{\circ}} := \mathcal{A}_d^{\tilde{\circ}} \cup \{\text{KonvertierePlanletInAktion}(mp, d, (m^s, m^{ab}))\}$ 
6: end for
7: // Selektiere Aktionen aus  $\mathcal{A}_d^{\tilde{\circ}}$  und berechne deren partielle Ordnung, um alle
8: // Managementannotationen in  $d$  auszuführen
9:  $(\mathcal{A}, \mathcal{O}) := \text{GenerierePartielleOrdnung}(d, \mathcal{A}_d^{\tilde{\circ}})$ 
10: // Generiere imperatives Managementmodell aus der partiellen Ordnung der Aktionen
11:  $\text{plan} := \text{GeneriereImperativesManagementmodell}(\mathcal{A}, \mathcal{O})$ 
12: return plan
```

Die generelle Funktionsweise kann wie folgt zusammengefasst werden: Ausgehend von einem Startzustand, welcher durch die Instanzinformationen des DMMN-Modells definiert ist, berechnet der Algorithmus mittels AI-Planning alle benötigten Planlets sowie deren Ausführungsreihenfolge zur Erreichung eines bestimmten Zielzustandes, der durch die erfolgreiche Abarbeitung aller im DMMN-Modell spezifizierten Managementannotationen definiert ist. Abschließend wird aus der erzeugten Ordnung der selektierten Planlets ein ausführbares imperatives Managementmodell generiert, beispielsweise ein Managementplan.

Im ersten Schritt werden dazu alle zum DMMN-Modell d statisch kompatiblen Planlets berechnet, welche strukturell und aufgabenbezogen kompatible Kandidaten zur Ausführung der Annotationen im DMMN-Modell darstellen (Zeile 2). Die Kandidatenmenge kann daher auch Planlets beinhalten, die bei der späteren Orchestrierung nicht berücksichtigt werden, da deren Annotationen von anderen Planlets ausgeführt werden. Die zugehörige Subroutine wird in [Abschnitt 5.4.4](#) vorgestellt. Anschließend werden alle Kandidaten-Planlets in sogenannte *Aktionen* konvertiert, welche jeweils ein bestimmtes Planlet hinsichtlich einer statisch kompatiblen Zuordnung an Elemente des DMMN-Modells bindet (Zeilen 4–6). Eine Aktion ist dabei ein 5-Tupel (siehe [Abschnitt 5.4.6](#)),

das die Anwendung eines Management-Planlets auf einen bestimmten Teil eines DMMN-Modells repräsentiert und dazu die ungebundenen Vorbedingungen und Effekte des Planlets an die Eigenschaften- und Annotationenelemente der zugehörigen im DMMN-Modell repräsentierten Komponenten und Relationen bindet. Eine Aktion ist somit eine individuelle Instanz eines Management-Planlets bezüglich eines bestimmten Teilgraphen eines DMMN-Modells. Folglich enthält die Menge $\mathfrak{A}_d^{\approx}$ alle statisch kompatiblen Aktionen, welche aus statisch kompatiblen Management-Planlets und deren Zuordnungen erzeugt werden. Durch die Konvertierung in Aktionen schwindet der Unterschied zwischen streng typisierten und generischen Management-Planlets, da beide Planlet-Typen durch die Instanziierung in eigenständige Aktionen an konkrete Topologieelemente des DMMN-Modells gebunden werden und sich die Generizität generischer Planlets dadurch auflöst. Die zugehörige Subroutine wird in [Abschnitt 5.4.6](#) vorgestellt.

Zur Ausführung aller Annotationen in d wird anschließend durch Aufruf der Subroutine *GenerierePartielleOrdnung* in Zeile 9 eine Menge geeigneter Aktionen $\mathcal{A}$ sowie deren Ausführungsreihenfolge berechnet, welche in Form einer partiellen Ordnung $\mathcal{O}$ spezifiziert wird (siehe [Algorithmus 5.11](#)). Gibt es mehrere mögliche Lösungen, wird nicht-deterministisch eine davon ausgewählt. Existiert keine Lösung, bricht die Subroutine mit einem Fehler ab.

Im letzten Schritt des Algorithmus werden die selektierten Aktionen und deren partielle Ordnung in Zeile 11 einer Subroutine übergeben, welche diese in ein ausführbares imperatives Managementmodell übersetzt (siehe [Abschnitt 5.5.2](#)). Die Subroutine kann für beliebige imperative Technologien implementiert werden, beispielsweise für Sprachen der Workflow-Technologie in Form von BPEL, BPMN oder BPMN4TOSCA. In der vorliegenden Arbeit wird die Umsetzung anhand der Workflow-Sprache BPEL gezeigt und die Routine im Rahmen der prototypischen Realisierung diesbezüglich implementiert (siehe [Kapitel 8](#)).

Die im Folgenden vorgestellten Subroutinen treffen an markierten Stellen nicht-deterministische Entscheidungen. Diese werden im Rahmen der prototypischen Implementierung (siehe [Kapitel 8](#)) durch das Konzept der Tiefensuche (engl. „Depth-First Search“) und des „Backtrackings“ [[CLRS01](#)] in deterministische Anweisungen überführt. Die daraus resultierende exponentielle Berechnungskomplexität wird in [Abschnitt 5.4.5.3](#) sowie [Abschnitt 5.6.4](#) diskutiert.

5.4.4 Algorithmus zur Berechnung statisch kompatibler Management-Planlets

In Zeile 2 des [Algorithmus 5.1](#) zur Transformation deklarativer DMMN-Modelle in imperative Managementmodelle wird eine Subroutine aufgerufen, die alle zu einem DMMN-Modell d statisch kompatiblen Management-Planlets inklusive deren statisch kompatiblen Zuordnungen berechnet. Die Subroutine ist in [Algorithmus 5.2](#) dargestellt und berechnet zunächst für alle Management-Planlets $mp \in MP$ die Menge der statisch kompatiblen Zuordnungen $\mathfrak{M}_{tf_{mp},d}^{\tilde{\circ}}$ bezüglich des übergebenen DMMN-Modells d (Zeile 4), wofür die Subroutine *BerechneKompatibleZuordnungen* $^{\tilde{\circ}}$ aufgerufen wird (siehe [Abschnitt 5.4.5](#)). Anschließend werden die berechneten statisch kompatiblen Zuordnungen jeweils als Paar mit dem Management-Planlet in die Menge der zu d statisch kompatiblen Management-Planlets $MP_d^{\tilde{\circ}}$ aufgenommen (Zeilen 7–9).

Algorithmus 5.2 BerechneStatischKompatiblePlanlets(d)

```

1:  $MP_d^{\tilde{\circ}} := \emptyset$ 
2: // Berechne für alle Planlets statisch kompatible Zuordnungen, falls diese existieren
3: for all ( $mp \in MP$ ) do
4:    $\mathfrak{M}_{tf_{mp},d}^{\tilde{\circ}} := \text{BerechneKompatibleZuordnungen}^{\tilde{\circ}}(tf_{mp}, d)$ 
5:   // Füge alle statisch kompatiblen Zuordnungen inklusive assoziiertem Planlet in
6:   // die Menge der statisch kompatiblen Management-Planlets hinzu
7:   for all ( $(m^s, m^{ab}) \in \mathfrak{M}_{tf_{mp},d}^{\tilde{\circ}}$ ) do
8:      $MP_d^{\tilde{\circ}} := MP_d^{\tilde{\circ}} \cup \{(mp, (m^s, m^{ab}))\}$ 
9:   end for
10: end for
11: return  $MP_d^{\tilde{\circ}}$ 

```

Die berechnete Menge der statisch kompatiblen Planlets $MP_d^{\tilde{\circ}}$ beinhaltet möglicherweise mehrere Planlets, welche dieselben Annotationen bezüglich einzelner Topologieelemente des DMMN-Modells ausführen. $MP_d^{\tilde{\circ}}$ stellt daher lediglich eine Kandidatenmenge möglicher Management-Planlets dar, aus welcher bei der späteren Selektion nicht notwendigerweise alle Planlets Verwendung finden.

5.4.5 Algorithmus zur Berechnung statisch kompatibler Zuordnungen

In diesem Abschnitt wird die Subroutine *BerechneKompatibleZuordnungen* ^{$\tilde{\circ}$} vorgestellt, die in Zeile 4 von [Algorithmus 5.2](#) aufgerufen wird, um alle statisch kompatiblen Zuordnungen zwischen einem Management-Planlet $mp \in MP$ und einem DMMN-Modell $d \in D$ zu berechnen. Die Routine wird unabhängig von Management-Planlets ausschließlich auf Basis von DMMN-Topologiefragmenten und DMMN-Modellen definiert, da der Algorithmus auch in [Kapitel 6](#) bei der Automatisierung von Managementmustern Verwendung findet. [Algorithmus 5.3](#) beschreibt die einzelnen Schritte der Routine in Pseudocode.

Algorithmus 5.3 BerechneKompatibleZuordnungen ^{$\tilde{\circ}$} (tf, d)

```

1:  $\mathcal{M}_{tf,d}^{\tilde{\circ}} := \emptyset$ 
2:  $\mathcal{M}_{tf,d}^{\tilde{s}} := \emptyset$ 
3: // Berechne alle strukturell kompatiblen Zuordnungen zwischen  $tf$  und  $d$ 
4: BerechneKompatibleZuordnungen $\tilde{s}$ ( $tf, d, \emptyset, \mathcal{M}_{tf,d}^{\tilde{s}}$ )
5: // Berechne für jede strukturell kompatible Zuordnung alle aufgabenbezogen kom-
6: // patiblen Zuordnungen und füge diese als statisch kompatible Zuordnungen hinzu
7: for all ( $m_i^s \in \mathcal{M}_{tf,d}^{\tilde{s}}$ ) do
8:   if ( $MA_{tf} \neq \emptyset$ ) then
9:      $\mathcal{M}_{tf,d,m_i^s}^{\tilde{ab}} := \emptyset$ 
10:    BerechneKompatibleZuordnungen $\tilde{ab}$ ( $tf, d, m_i^s, \emptyset, \mathcal{M}_{tf,d,m_i^s}^{\tilde{ab}}$ )
11:    for all ( $m_j^{ab} \in \mathcal{M}_{tf,d,m_i^s}^{\tilde{ab}}$ ) do
12:       $\mathcal{M}_{tf,d}^{\tilde{\circ}} := \mathcal{M}_{tf,d}^{\tilde{\circ}} \cup \{(m_i^s, m_j^{ab})\}$ 
13:    end for
14:  else
15:    // Falls  $tf$  keine Annotationen spezifiziert, dann definiert sich die statische
16:    // Kompatibilität ausschließlich durch die strukturelle Kompatibilität
17:     $\mathcal{M}_{tf,d}^{\tilde{\circ}} := \mathcal{M}_{tf,d}^{\tilde{\circ}} \cup \{(m_i^s, \emptyset)\}$ 
18:  end if
19: end for
20: return  $\mathcal{M}_{tf,d}^{\tilde{\circ}}$ 

```

Das Ziel des Algorithmus ist die Berechnung der in Zeile 1 deklarierten Menge $\mathfrak{M}_{tf,d}^{\tilde{s}}$, welche alle statisch kompatiblen Zuordnungen der Elemente eines Topologiefragments tf zu einem DMMN-Modell d beinhaltet, d. h. Paare (m^s, m^{ab}) . Im ersten Schritt werden zunächst alle strukturell kompatiblen Zuordnungen zwischen tf und d berechnet. Dazu wird in Zeile 2 die Menge $\mathfrak{M}_{tf,d}^{\tilde{s}}$ deklariert, welche durch die in [Abschnitt 5.4.5.1](#) eingeführte Subroutine *BerechneKompatibleZuordnungen* $^{\tilde{s}}$ berechnet wird (Zeile 4). Da eine statisch kompatible Zuordnung aus einem Paar (m^s, m^{ab}) besteht, wird für jede strukturell kompatible Zuordnung m_i^s aus $\mathfrak{M}_{tf,d}^{\tilde{s}}$ (Zeile 7) durch Aufruf der Subroutine *BerechneKompatibleZuordnungen* $^{\tilde{ab}}$ ([Abschnitt 5.4.5.2](#)) alle zu dieser Struktur aufgabenbezogen kompatiblen Zuordnungen berechnet (Zeile 10), falls das Topologiefragment tf Aufgaben in Form von Annotationen spezifiziert (Zeile 8). Die Menge $\mathfrak{M}_{tf,d,m_i^s}^{\tilde{ab}}$ (Zeile 9) beinhaltet somit alle aufgabenbezogen kompatiblen Zuordnungen zwischen tf und d bezüglich einer strukturellen Zuordnung m_i^s . Jede strukturell kompatible Zuordnung m_i^s und jede zu dieser aufgabenbezogen kompatible Zuordnung m_j^{ab} werden anschließend paarweise in die Ergebnismenge aufgenommen (Zeile 12). Deklariert tf keine Annotationen, bestehen diese Paare lediglich aus der strukturellen Zuordnung (Zeile 17).

5.4.5.1 Algorithmus zur Berechnung strukturell kompatibler Zuordnungen

Die Berechnung aller strukturell kompatiblen Zuordnungen zwischen einem Topologiefragment tf und einem DMMN-Modell d , d. h. die Berechnung der Menge $\mathfrak{M}_{tf,d}^{\tilde{s}}$, kann auf die Existenz von Subgraphisomorphismen zurückgeführt werden: Kann der durch tf aufgespannte Graph aus Komponenten- und Relationenelementen isomorph auf kompatible Elemente eines oder mehrerer Teilgraphen in d abgebildet werden, repräsentiert jeder dieser Subgraphisomorphismen eine strukturell kompatible Zuordnung (vgl. [Definition 64](#)). [Algorithmus 5.4](#) beschreibt deren Berechnung, welche auf dem VF- bzw. VF2-Algorithmus [[CFSV98](#); [CFSV99](#); [CFSV01](#); [CFSV04](#)] basiert und diesen für DMMN adaptiert. Die Berechnung erfordert im ungünstigsten Fall exponentielle Laufzeit, die sich inhärent aus der Komplexität des Entscheidungsproblems zur Bestimmung von Subgraphisomorphismen ergibt (siehe [Abschnitt 5.4.5.3](#)).

Der Algorithmus berechnet jede Zuordnung m^s schrittweise und übernimmt diese in die Ergebnismenge $\mathfrak{M}_{tf,d}^{\approx_s}$, sobald alle Topologieelemente in tf abgebildet sind (Zeilen 3–5). Dabei wird in jedem Schritt eine Menge möglicher Kandidaten berechnet (Zeile 8), die potenziell in die aktuelle, anfangs leere, Zuordnung m^s aufgenommen werden kann (siehe Algorithmus 5.5). Um alle möglichen Zuordnungen zu finden, wird für jeden Kandidaten eine Kopie $m^{s'}$ erstellt und die Berechnung rekursiv für diese fortgesetzt (Zeile 17). Handelt es sich beim Kandidaten um eine Relation, wird auch deren Ursprungs- und Zielkomponente in die Kopie aufgenommen (Zeilen 13–15). In jedem Schritt gilt, dass alle durch m^s zugeordneten Elemente einen korrekten Subgraphisomorphismus zwischen einem Teilgraphen in tf und einem Teilgraphen in d darstellen.

Algorithmus 5.4 BerechneKompatibleZuordnungen $^{\approx_s}(tf, d, m^s, \mathfrak{M}_{tf,d}^{\approx_s})$

```

1: // Wenn alle Topologieelemente in  $tf$  durch  $m^s$  strukturell kompatibel auf einen
2: // Teilgraphen in  $d$  abgebildet werden, füge  $m^s$  in die Resultatsmenge hinzu
3: if  $(\forall te_i \in TE_{tf} \exists! te_t \in TE_d : (te_i, te_t) \in m^s)$  then
4:    $\mathfrak{M}_{tf,d}^{\approx_s} := \mathfrak{M}_{tf,d}^{\approx_s} \cup \{m^s\}$ 
5: else
6:   // Berechne Topologieelemente, von denen jedes in Kombination mit  $m^s$  einen
7:   // strukturell kompatiblen Subgraphisomorphismus zwischen  $tf$  und  $d$  darstellt
8:   kandidaten := BerechneKandidaten $^{\approx_s}(tf, d, m^s)$ 
9:   // Kopiere  $m^s$  für jeden Kandidaten und führe die Berechnung rekursiv fort
10:  for all  $((te_{tf}, te_d) \in \text{kandidaten})$  do
11:     $m^{s'} := m^s \cup \{(te_{tf}, te_d)\}$ 
12:    // Für Kanten auch die zugehörigen Komponentenelemente einfügen
13:    if  $(te_{tf} \in R_{tf} \wedge te_d \in R_d)$  then
14:       $m^{s'} := m^{s'} \cup \{(\pi_3(te_{tf}), \pi_3(te_d)), (\pi_4(te_{tf}), \pi_4(te_d))\}$ 
15:    end if
16:    // Rekursiver Selbstaufzuruf zur Fortführung der Berechnung
17:    BerechneKompatibleZuordnungen $^{\approx_s}(tf, d, m^{s'}, \mathfrak{M}_{tf,d}^{\approx_s})$ 
18:  end for
19: end if

```

Die Berechnung einer strukturell kompatiblen Zuordnung m^s wird daher durch einen kontinuierlich wachsenden Subgraphisomorphismus zwischen einem Teilgraphen in tf und einem Teilgraphen in d realisiert, bis tf vollständig abgebildet ist. Die Berechnung potenzieller Kandidaten, welche in jedem Schritt in m^s aufgenommen werden können, erfordert daher eine explizite Analyse, ob deren Aufnahme einen korrekten Subgraphisomorphismus zwischen den jeweiligen Teilgraphen darstellen würde.

Die Berechnung möglicher Kandidaten zur Aufnahme in m^s wird von [Algorithmus 5.5](#) durchgeführt. Die grundsätzliche Funktionsweise kann wie folgt zusammengefasst werden: Da m^s zu jedem Zeitpunkt einen korrekten Subgraphisomorphismus zwischen einem Teilgraphen in tf und einem Teilgraphen in d darstellt, werden für die bisher durch m^s zugeordneten Komponentenelemente (Zeile 12) alle ein- und ausgehenden Kanten, die noch nicht durch m^s abgebildet sind, auf Kompatibilität untersucht (Zeilen 13–15) und im positiven Falle in die Kandidatenmenge aufgenommen (Zeile 20 bzw. Zeile 28). Für ausgehende Kanten wird dabei überprüft, ob (i) das Zielkomponentenelement der Kante des Topologiefragments kompatibel zum Zielkomponentenelement der zu vergleichenden Kante des DMMN-Modells ist (Zeile 17). Zudem wird überprüft, ob (ii) die beiden Zielkomponentenelemente der zu vergleichenden Kanten schon durch m^s zugeordnet sind (Zeilen 18–19). In letzterem Fall muss das zugeordnete Element dem Ziel der zu vergleichenden Kante entsprechen. Dadurch können Kanten zwischen bereits zugeordneten Komponentenelementen in m^s aufgenommen werden. Die Kompatibilität eingehender Kanten wird entsprechend überprüft (Zeilen 25–31). Da m^s anfangs keine Zuordnungen enthält, werden beim ersten Aufruf des Algorithmus alle kompatiblen Komponentenelemente beider Graphen als Kandidaten erfasst und diese als Ergebnis zurückgeliefert (Zeilen 4–9). Durch dieses Vorgehen wächst der Teilgraph in tf , welcher durch m^s auf einen Teilgraphen in d abgebildet wird, sukzessive, bis alle Topologeelemente in tf vollständig durch einen Subgraphisomorphismus *strukturell kompatibel* auf Elemente in d abgebildet sind.

Algorithmus 5.5 BerechneKandidaten $\tilde{\approx}^s(tf, d, m^s)$

```
1: kandidaten :=  $\emptyset$ 
2: // Falls keine strukturelle Zuordnung existiert, liefere alle kompatiblen Paare aus
3: // Komponentenelementen des Planlets und Komponentenelementen des DMMN-Modells
4: if ( $m^s = \emptyset$ ) then
5:   for all ( $k_{tf} \in K_{tf}, k_d \in K_d \mid k_{tf} \tilde{\approx}_{te} k_d$ ) do
6:     kandidaten := kandidaten  $\cup \{(k_{tf}, k_d)\}$ 
7:   end for
8:   return kandidaten
9: end if

10: // Sonst überprüfe alle bisher zugeordneten Komponentenelemente bezüglich der
11: // Kompatibilität aller ein- und ausgehenden Kanten, die noch nicht abgebildet sind
12: for all ( $(k_{tf} \in K_{tf}, k_d \in K_d) \in m^s$ ) do
13:   for all ( $r_{tf} \in R_{tf} \mid \nexists r_x \in R_d: (r_{tf}, r_x) \in m^s$ ) do
14:     for all ( $r_d \in R_d \mid \nexists r_y \in R_{tf}: (r_y, r_d) \in m^s$ ) do
15:       if ( $r_{tf} \tilde{\approx}_{te} r_d$ ) then
16:         // Überprüfe Zielkomponenten für ausgehende Kanten
17:         if ( $\pi_3(r_{tf}) = k_{tf} \wedge \pi_3(r_d) = k_d \wedge \pi_4(r_{tf}) \tilde{\approx}_{te} \pi_4(r_d)$ ) then
18:           if ( $\exists k_x \in K_{tf}: (k_x, \pi_4(r_d)) \in m^s \Rightarrow k_x = \pi_4(r_{tf})$ ) then
19:             if ( $\exists k_y \in K_d: (\pi_4(r_{tf}), k_y) \in m^s \Rightarrow k_y = \pi_4(r_d)$ ) then
20:               kandidaten := kandidaten  $\cup \{(r_{tf}, r_d)\}$ 
21:             end if
22:           end if
23:         end if
24:         // Überprüfe Ursprungskomponenten für eingehende Kanten
25:         if ( $\pi_4(r_{tf}) = k_{tf} \wedge \pi_4(r_d) = k_d \wedge \pi_3(r_{tf}) \tilde{\approx}_{te} \pi_3(r_d)$ ) then
26:           if ( $\exists k_x \in K_{tf}: (k_x, \pi_3(r_d)) \in m^s \Rightarrow k_x = \pi_3(r_{tf})$ ) then
27:             if ( $\exists k_y \in K_d: (\pi_3(r_{tf}), k_y) \in m^s \Rightarrow k_y = \pi_3(r_d)$ ) then
28:               kandidaten := kandidaten  $\cup \{(r_{tf}, r_d)\}$ 
29:             end if
30:           end if
31:         end if
32:       end if
33:     end for
34:   end for
35: end for
36: return kandidaten
```

5.4.5.2 Algorithmus zur Berechnung aufgabenbezogener Zuordnungen

In diesem Abschnitt wird die Berechnung aller aufgabenbezogen kompatiblen Zuordnungen zwischen einem Topologiefragment tf und einem DMMN-Modell d bezüglich einer strukturell kompatiblen Zuordnung m^s beschrieben. [Algorithmus 5.6](#) beschreibt die Berechnung in Pseudocode und wird von [Algorithmus 5.3](#) in Zeile 10 als Subroutine aufgerufen, um $\mathfrak{M}_{tf,d,m^s}^{\tilde{ab}}$ zu bestimmen.

Algorithmus 5.6 BerechneKompatibleZuordnungen $^{\tilde{ab}}(tf, d, m^s, m^{ab}, \mathfrak{M}_{tf,d,m^s}^{\tilde{ab}})$

```

1: // Wenn alle Annotationenelemente in  $tf$  durch  $m^{ab}$  aufgabenbezogen kompatibel
2: // auf Annotationen in  $d$  abgebildet werden, füge  $m^{ab}$  in die Resultatsmenge hinzu
3: if  $(\forall ma_i \in MA_{tf} \exists! ma_t \in MA_d : (ma_i, ma_t) \in m^{ab})$  then
4:   if  $(\forall (ma_{d_1}, ma_{d_2}, C) \in P_d : (\exists (ma_1, ma_{d_1}) \in m^{ab} \vee \exists (ma_2, ma_{d_2}) \in m^{ab}) \Rightarrow$ 
5:      $(\exists (ma_{tf_1}, ma_{d_1}) \in m^{ab} \wedge \exists (ma_{tf_2}, ma_{d_2}) \in m^{ab} \wedge (z = * \vee z = C) \wedge$ 
6:      $((ma_{tf_1}, ma_{tf_2}, z) \in P_{tf} \vee (ma_{tf_2}, ma_{tf_1}, z) \in P_{tf})))$  then
7:      $\mathfrak{M}_{tf,d,m^s}^{\tilde{ab}} := \mathfrak{M}_{tf,d,m^s}^{\tilde{ab}} \cup \{m^{ab}\}$ 
8:   end if
9: else
10:  // Erfasse für jedes abgebildete Topologeelement alle kompatiblen Management-
11:  // annotationen, die noch nicht durch  $m^{ab}$  zugeordnet sind, als Kandidaten
12:  kandidaten :=  $\emptyset$ 
13:  for all  $((te_{tf}, te_d) \in m^s)$  do
14:     $MA_{te_{tf}} := \text{annotationen}_{tf}(te_{tf})$ 
15:     $MA_{te_d} := \text{annotationen}_d(te_d)$ 
16:    for all  $(ma_i \in MA_{te_{tf}} \mid \nexists ma_p \in MA_{te_d} : (ma_i, ma_p) \in m^{ab})$  do
17:      for all  $(ma_j \in MA_{te_d} \mid \nexists ma_q \in MA_{te_{tf}} : (ma_q, ma_j) \in m^{ab})$  do
18:        if  $(\text{AnnotationenKompatibel}(ma_i, ma_j, tf, d, m^{ab}))$  then
19:          kandidaten := kandidaten  $\cup \{(ma_i, ma_j)\}$ 
20:        end if
21:      end for
22:    end for
23:  end for
24:  // Kopiere  $m^{ab}$  für jeden Kandidaten und führe die Berechnung rekursiv fort
25:  for all  $((ma_{tf}, ma_d) \in \text{kandidaten})$  do
26:     $m^{ab'} := m^{ab} \cup \{(ma_{tf}, ma_d)\}$ 
27:    BerechneKompatibleZuordnungen $^{\tilde{ab}}(tf, d, m^s, m^{ab'}, \mathfrak{M}_{tf,d,m^s}^{\tilde{ab}})$ 
28:  end for
29: end if

```

Ähnlich zur Berechnung von Subgraphisomorphismen im vorangegangenen Abschnitt handelt es sich um einen rekursiven Algorithmus, der die Zuordnungen schrittweise berechnet und als Eingabe (i) das Topologiefragment tf , (ii) das DMMN-Modell d , (iii) die strukturelle Zuordnung m^s , für welche die aufgabenbezogene Kompatibilität berechnet werden soll, sowie (iv) die sukzessiv wachsende aufgabenbezogene Zuordnung m^{ab} und (v) die Ergebnismenge $\mathcal{M}_{tf,d,m^s}^{ab}$ übermittelt bekommt. Durch jeden Aufruf wird jeweils ein neues Paar bestehend aus einer Annotation aus tf und einer kompatiblen Annotation aus d zu m^{ab} hinzugefügt. Ist jede Managementannotation aus tf eindeutig durch m^{ab} auf eine Annotation in d abgebildet und werden alle gleichzeitigen Ausführungen von Annotationen eingehalten (ab Zeile 3), wird die Zuordnung m^{ab} in die Ergebnismenge aufgenommen (Zeile 7). Falls nicht, werden alle Managementannotationen zweier kompatibler Topologieelemente aus m^s (Zeile 13), welche noch nicht in m^{ab} enthalten sind (Zeilen 14–17), miteinander auf Kompatibilität verglichen (Zeile 18) und im positiven Fall in eine Kandidatenmenge aufgenommen (Zeile 19). Anschließend wird für jeden Kandidaten eine Kopie $m^{ab'}$ erstellt und die Berechnung rekursiv fortgesetzt (Zeilen 25–28), bis alle möglichen aufgabenbezogen kompatiblen Zuordnungen berechnet sind.

Zur Verifikation der Kompatibilität zweier Annotationen wird in Zeile 18 **Algorithmus 5.7** aufgerufen. Diese Verifikation besteht dabei nicht nur aus der Überprüfung der Kompatibilität der beiden Managementannotationen selbst, sondern auch aus der Überprüfung, ob alle im DMMN-Modell diesbezüglich deklarierten Reihenfolgespezifikationen vom Topologiefragment berücksichtigt werden. **Algorithmus 5.7** bekommt dazu die zu überprüfenden Annotationen ma_{tf} und ma_d , deren DMMN-Modelle tf und d sowie die noch unvollständige Zuordnung m^{ab} als Eingabe. Der Algorithmus evaluiert zunächst in Zeile 2, ob die beiden Managementannotationen bezüglich Typ und Konfiguration zueinander kompatibel sind (siehe **Definition 61**). Führt das Management-Planlet mehr als eine Managementannotation des DMMN-Modells aus, müssen die im DMMN-Modell festgelegten Reihenfolgespezifikationen bezüglich dieser Annotationen vom Planlet berücksichtigt werden. Dazu erstellt der Algorithmus zunächst eine lokale Kopie von m^{ab} (Zeile 4), in welche die aufgabenbezogene

Zuordnung der zu überprüfenden Annotationen testweise aufgenommen wird. Im Folgenden wird für diese Zuordnung $m^{ab'}$ überprüft, ob alle Reihenfolgespezifikationen bezüglich der betroffenen Managementannotationen vom Planlet eingehalten werden. Falls im DMMN-Modell d eine Reihenfolgespezifikation festgelegt ist, die das Topologiefragment tf des Planlets nicht einhält, sind die zu überprüfenden Annotationen ma_{tf} und ma_d nicht kompatibel.

Dazu wird zunächst für jede Reihenfolgespezifikation in d überprüft (Zeile 7), ob das Management-Planlet die betroffenen Annotationen ausführt, d. h. ob es eine Zuordnung zweier Annotationen ma_{tf_1} und ma_{tf_2} des Topologiefragments auf die Annotationen ma_{d_1} und ma_{d_2} dieser Reihenfolgespezifikationen gibt (Zeile 10). Falls es eine solche Zuordnung gibt, muss das Planlet die Reihenfolgespezifikation erfüllen. Dazu wird in Zeile 14 zunächst überprüft, ob (i) die Reihenfolgespezifikation obsolet ist (vgl. Definition 49) oder (ii) ob das Planlet jegliche Reihenfolge bezüglich ma_{d_1} und ma_{d_2} unterstützt, was durch die Existenz einer Reihenfolgespezifikation des Typs $*$ auf Seite des Planlets ausgedrückt wird. Falls dies nicht zutrifft, wird im Folgenden überprüft, ob das Planlet exakt die Reihenfolgespezifikation des DMMN-Modells unterstützt: Ist die Ausführungsreihenfolge der Annotationen ma_{d_1} und ma_{d_2} vorgegeben, muss das Planlet dieselbe Ausführungsreihenfolge für die entsprechenden Annotationen ma_{tf_1} und ma_{tf_2} zusichern (Zeile 18). Spezifiziert das DMMN-Modell, dass der Effekt der Ausführung beider Annotationen ma_{d_1} und ma_{d_2} einer gleichzeitigen Ausführung entsprechen muss, erfordert dies eine diesbezüglich semantisch äquivalente Zusicherung seitens des Management-Planlets (Zeile 26). Da diese Semantik auf zwei unterschiedliche Arten modelliert werden kann, muss das Topologiefragment des Management-Planlets eine der beiden Möglichkeiten gewährleisten (Zeile 24 bzw. Zeile 25).

Falls eine dieser Bedingungen nicht erfüllt ist, sind die Managementannotationen nicht kompatibel und der Algorithmus liefert einen negativen Wahrheitswert zurück. Die Routine sichert durch dieses Verfahren zu, dass nur Kandidaten vorgeschlagen werden, welche in m^{ab} aufgenommen werden können ohne die im DMMN-Modell spezifizierten Reihenfolgen zu verletzen.

Algorithmus 5.7 AnnotationenKompatibel($ma_{tf}, ma_d, tf, d, m^{ab}$)

```
1: // Überprüfung der Kompatibilität der zu vergleichenden Annotationen selbst
2: if ( $ma_{tf} \approx_{ma} ma_d$ ) then
3:   // Falls kompatibel, nehme die aufgabenbezogene Zuordnung testweise auf
4:    $m^{ab'} := m^{ab} \cup \{(ma_{tf}, ma_d)\}$ 
5:   // Überprüfe für alle Reihenfolgespezifikationen in  $d$ , ob diese von  $tf$ 
6:   // eingehalten werden, falls das Planlet beide Annotationen ausführt
7:   for all  $((ma_{d_1}, ma_{d_2}, z) \in P_d)$  do
8:     // Falls beide Annotationen einer Reihenfolgespezifikation durch  $m^{ab}$  zu-
9:     // geordnet sind, überprüfe, ob das Planlet diese Reihenfolge einhält
10:    if  $(\exists(ma_{tf_1}, ma_{d_1}) \in m^{ab'} \wedge \exists(ma_{tf_2}, ma_{d_2}) \in m^{ab'})$  then
11:      // Falls die Reihenfolgespezifikation des DMMN-Modells obsolet ist,
12:      // d. h.  $z = *$ , oder das Topologiefragment alle Reihenfolgen für  $ma_{d_1}$ 
13:      // und  $ma_{d_2}$  unterstützt, wird die Reihenfolgespezifikation nicht verletzt
14:      if  $(z \neq * \wedge \nexists(ma_{tf_1}, ma_{tf_2}, *) \in P_{tf} \wedge \nexists(ma_{tf_2}, ma_{tf_1}, *) \in P_{tf})$  then
15:        // Falls die Reihenfolgespezifikation des DMMN-Modells eine Ordnung
16:        // festlegt, muss das Topologiefragment diese einhalten. Andernfalls
17:        // wird die Reihenfolgespezifikation verletzt
18:        if  $(z = O \wedge \nexists(ma_{tf_1}, ma_{tf_2}, O) \in P_{tf})$  then
19:          return false
20:        end if
21:      // Falls die Reihenfolgespezifikation des DMMN-Modells die gleich-
22:      // zeitige Ausführung der Annotationen festlegt, muss das Topologie-
23:      // fragment diese einhalten. Andernfalls wird die Reihenfolge verletzt
24:       $b_1 = \nexists(ma_{tf_1}, ma_{tf_2}, C) \in P_{tf}$ 
25:       $b_2 = \nexists(ma_{tf_2}, ma_{tf_1}, C) \in P_{tf}$ 
26:      if  $(z = C \wedge b_1 \wedge b_2)$  then
27:        return false
28:      end if
29:    end if
30:  end if
31: end for
32: else
33:   return false
34: end if
35: // Falls die Annotationen selbst kompatibel sind und keine Reihenfolgespezifikationen
36: // verletzt wurden, sind diese auch im Kontext des DMMN-Modells kompatibel
37: return true
```

5.4.5.3 Diskussion zur Berechnung statisch kompatibler Zuordnungen

Die Laufzeit von [Algorithmus 5.3](#), welche sich inhärent aus dem Berechnungsaufwand von Subgraphisomorphismen ergibt und im schlechtesten Fall exponentiellem Aufwand entspricht [[CFSV04](#)], beeinflusst die Berechnung statisch kompatibler Zuordnungen nur bedingt. Zum einen sind Topologiefragmente von Planlets typischerweise kleine Graphen, welche aus zwei bis drei Komponenten und Relationen bestehen. Dies resultiert aus der gewünschten Wiederverwendbarkeit von Planlets für verschiedene Anwendungen: Je größer das Topologiefragment, desto eingeschränkter ist dessen Anwendungsbereich, da Teile der betroffenen Anwendungen exakt derselben Struktur entsprechen müssen. Somit ist eine grundsätzliche Zielsetzung, die Wiederverwendbarkeit von Planlets auf ein Maximum zu heben, indem Management-Planlets im Idealfall nur eine einzige Managementaufgabe ausführen. Im Gegensatz zur Komposition mehrerer Managementaufgaben durch ein einzelnes Planlet führt der oben beschriebene Ansatz zu kleineren Topologiefragmenten und somit in der Praxis zu vernachlässigbaren Auswirkungen der theoretisch hohen Laufzeit.

Durch die Aufteilung der Funktionalität auf mehrere Planlets reduzieren sich auch die Abhängigkeiten der einzelnen Planlets zu den Relationen der Komponenten, was zu dünnen Topologiefragmenten (engl. „*sparse graphs*“) führt. Dies wirkt sich ebenfalls positiv auf die Laufzeit des Algorithmus aus. Des Weiteren führt die Domäne aufgrund der Heterogenität bezüglich Komponenten, Diensten und Managementtechnologien (vgl. [Abschnitt 2.2.3.1](#)) zu Topologiefragmenten, welche untereinander nur eine minimale Anzahl isomorpher Teilgraphen aufweisen. Die Installation eines Webserverns auf Linux- bzw. Windows-basierten Betriebssystemen durch verschiedene Management-Planlets führt beispielsweise zu Topologiefragmenten, welche lediglich die Komponente des Webserverns identisch in beiden Fragmenten enthalten, sich jedoch schon durch die unterliegende Betriebssystemkomponente unterscheiden. Der Algorithmus kann die Berechnung möglicher strukturell kompatibler Zuordnungen somit in vielen Fällen schon nach wenigen Schritten abschließen, wodurch die theoretische exponentielle Laufzeit in der Praxis keine Relevanz erfährt. Die Optimierung der Laufzeit ist daher nicht Gegenstand der vorliegenden Arbeit.

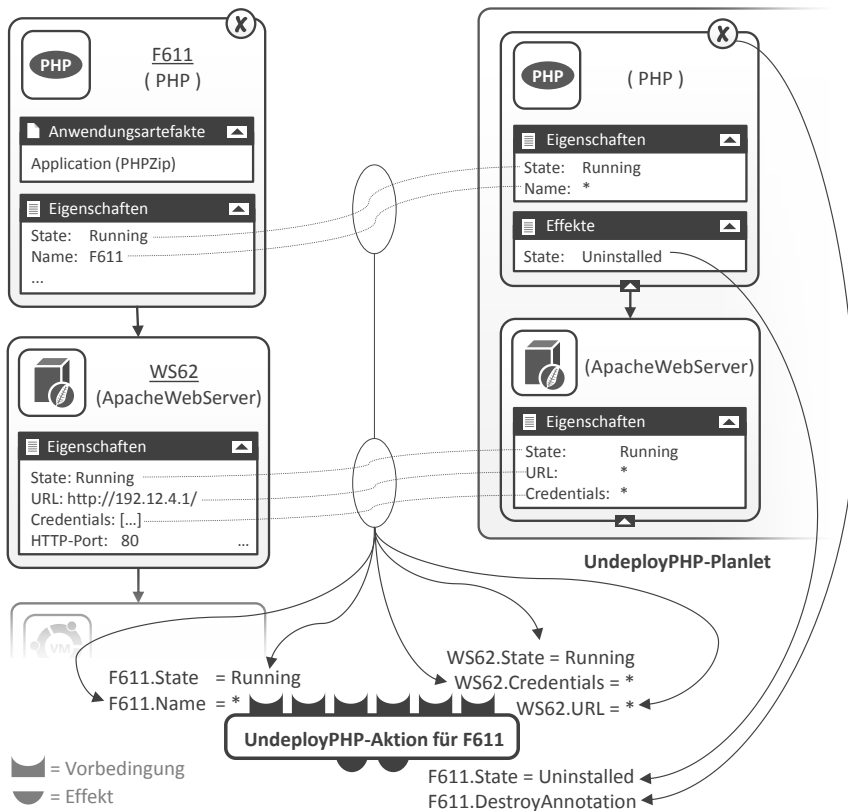


Abbildung 5.11: Illustration des Algorithmus zur Planlet-Konvertierung

5.4.6 Algorithmus zur Konvertierung von Management-Planlets in Aktionen

In Zeile 5 des [Algorithmus 5.1](#) zur Transformation deklarativer DMMN-Modelle in imperative Managementmodelle werden alle statisch kompatiblen Planlets in $MP_d^{\approx}$ bezüglich jeder statisch kompatiblen Zuordnung zum DMMN-Modell in jeweils eine *Aktion* konvertiert. Eine Aktion stellt die Instanziierung eines Planlets dar, welche fest an Elemente des DMMN-Modells gebunden ist. [Abbildung 5.11](#) zeigt dieses Prinzip anhand eines simplifizierten Beispiels, das Vorbedingungen und Effekte bezüglich Relationen auslöst. Auf der linken Seite

ist ein Ausschnitt eines DMMN-Modells d abgebildet, das die Deinstallation der laufenden PHP-Anwendung „F611“ auf einem Apache Webserver „WS62“ deklariert. Die rechte Seite zeigt ein Planlet, das diese Deinstallation durchführen kann. Aufgrund der statischen Kompatibilität des Planlets zu d bezüglich der genannten Komponenten ist das Planlet in $MP_d^{\approx}$ enthalten. Bei der Konvertierung in eine Aktion werden alle Vorbedingungen, Effekte und Annotationen des Planlets an die jeweiligen Gegenparte des DMMN-Modells gebunden. Das Resultat der Konvertierung zeigt das untere Drittel in [Abbildung 5.11](#): Die erzeugte Aktion spezifiziert konkrete Vorbedingungen an Eigenschaften der Topologieelemente in d , beispielsweise, dass die Komponente „F611“ bezüglich der Eigenschaft *State* den Wert „Running“ haben muss. Zudem werden alle Effekte und Annotationen des Planlets in Effekte der Aktion konvertiert, welche ebenfalls an Topologieelemente in d gebunden sind. Effekte der Aktion betreffen ausschließlich die Komponente „F611“, welche in Zustand „Uninstalled“ überführt und deren Außerbetriebnahmeannotation ausgeführt wird.

[Algorithmus 5.8](#) beschreibt diese Konvertierung in Pseudocode und konvertiert ein Planlet mp bezüglich einer statisch kompatiblen Zuordnung (m^s, m^{ab}) zu einem DMMN-Modell d in eine Aktion. Dazu werden über alle strukturell kompatiblen Topologieelemente (Zeilen 2–26) iteriert und folgende Konvertierungen durchgeführt: Für jedes Topologieelement $te_{tf_{mp}}$ im Topologiefragment des Planlets werden dessen Effekte mit dem strukturell kompatiblen Topologieelement des DMMN-Modells assoziiert und die resultierenden Tupel als Effekte zur Menge $Effekte_e$ hinzugefügt, welche alle Effekte der Aktion auf Eigenschaften beinhaltet (Zeilen 4–6). Gleichermaßen wird von Zeile 8 bis Zeile 10 über die Annotationenelemente des Topologieelements $te_{tf_{mp}}$ iteriert und für jedes das im DMMN-Modell zugeordnete Topologieelement in Kombination mit dem zugeordneten Annotationenelement als Paar in die Menge $Effekte_{ma}$ hinzugefügt, welche alle Effekte der Aktion bezüglich ausgeführter Managementannotationen beinhaltet. Die Vorbedingungen der Aktion werden durch die Menge Vor_e beschrieben. Diese werden von Zeile 12 bis Zeile 14 durch Iteration über die eigenen Eigenschaftenelemente des Topologieelements des Planlets generiert, indem jedes Eigenschaftenelement an das entsprechend

Algorithmus 5.8 KonvertierePlanletInAktion($mp, d, (m^s, m^{ab})$)

```
1:  $Vor_e, Effekte_e, Effekte_{ma} := \emptyset$ 
2: for all  $((te_{t_{f_{mp}}}, te_d) \in m^s)$  do
3:   // Konvertiere jeden Effekt des Planlets in einen Effekt der Aktion
4:   for all  $(ef_i \in effekte_{mp}(te_{t_{f_{mp}}}))$  do
5:      $Effekte_e := Effekte_e \cup \{(te_d, ef_i)\}$ 
6:   end for
7:   // Konvertiere alle Annotationen, die das Planlet ausführt, in Effekte der Aktion
8:   for all  $((ma_i, ma_t) \in m^{ab} \mid ma_i \in annotationen_{t_{f_{mp}}}(te_{t_{f_{mp}}}))$  do
9:      $Effekte_{ma} := Effekte_{ma} \cup \{(te_d, ma_t)\}$ 
10:  end for
11:  // Konvertiere jedes eigene Eigenschaftenelement in eine Vorbedingung der Aktion
12:  for all  $(e_j \in eigeneEigenschaften_{t_{f_{mp}}}(te_{t_{f_{mp}}}))$  do
13:     $Vor_e := Vor_e \cup \{(te_d, e_j)\}$ 
14:  end for
15:  // Konvertiere alle Eigenschaftsreferenzen der Ein- und Ausgabeparameter aller
16:  // vom Planlet ausgeführten Operationen in Vorbedingungen bzw. Effekte
17:  for all  $(i_k \in eigeneSchnittstellen_d(te_d))$  do
18:    for all  $(o_p \in operationen_d(i_k))$  do
19:       $eI := eigeneSchnittstellen_{t_{f_{mp}}}(te_{t_{f_{mp}}})$ 
20:      if  $(\exists i_r \in eI \exists o_w \in operationen_{t_{f_{mp}}}(i_r): i_r \approx_{i_k} i_k \wedge o_w \approx_{o_p} o_p)$  then
21:         $Vor_e := Vor_e \cup \{\text{KonvertiereEingaben}(o_p)\}$ 
22:         $Effekte_e := Effekte_e \cup \{\text{KonvertiereAusgaben}(o_p, Effekte_e)\}$ 
23:      end if
24:    end for
25:  end for
26: end for
27: return  $(Vor_e, Effekte_e, Effekte_{ma}, mp, (m^s, m^{ab}))$ 
```

zugeordnete Topologieelement des DMMN-Modells gebunden wird. Bei der Konvertierung gilt die Grundannahme, dass alle vom Planlet als Vorbedingung spezifizierten Managementoperationen ausgeführt werden, welche nicht zur Kompensation anderer Operationen verwendet werden (vgl. [Abschnitt 4.2.3.2](#)). Spezifizieren die entsprechenden Operationenelemente des DMMN-Modells

Eigenschaftsreferenzen für Ein- oder Ausgabeparameter, stellen diese somit Vorbedingungen bzw. Effekte der Planlet-Ausführung dar und müssen daher bei der Konvertierung in Aktionen berücksichtigt werden: Eine Aktion kann erst ausgeführt werden, wenn die referenzierten Eigenschaften der Eingabeparameter verfügbar sind. Referenzierte Eigenschaften der Ausgabeparameter werden vom Planlet ins Instanzmodell übertragen. **Algorithmus 5.8** iteriert daher von Zeile 17 bis Zeile 25 über alle eigenen Managementschnittstellen und zugehörigen Operationenelemente des Topologieelements te_d im DMMN-Modell und überprüft, welche dieser Operationen vom Planlet ausgeführt werden (Zeile 20): Spezifiziert das Topologieelement $te_{f_{mp}}$ des Planlets eine Vorbedingung in Form einer Operation, welche im DMMN-Modell vom zugeordneten Topologieelement te_d kompatibel spezifiziert wird, so führt das Planlet diese Operation aus und muss folglich deren Eingaben bereitstellen und Ausgaben verarbeiten. Daher werden Eigenschaftsreferenzen bezüglich der Eingabeparameter in die Menge der Vorbedingungen aufgenommen (Zeile 21) und Eigenschaftsreferenzen der Ausgabeparameter als Effekte erfasst (Zeile 22). Da Kompensationsoperationen keine Eigenschaftsreferenzen spezifizieren dürfen (vgl. **Definition 35**), werden keine falschen Vorbedingungen bzw. Effekte erzeugt. Eine Aktion ist somit ein 5-Tupel bestehend aus Vor_e , $Effekte_e$, $Effekte_{ma}$, mp und (m^s, m^{ab}) , vgl. Zeile 27.

Algorithmus 5.9 KonvertiereEingaben(o)

```

1:  $Vor_e := \emptyset$ 
2: for all ( $param_z \in \pi_3(o)$ ) do
3:    $er := \pi_2(param_z)$ 
4:   if ( $er \neq \perp$ ) then
5:      $Vor_e := Vor_e \cup \{(\pi_1(er), \pi_2(er))\}$ 
6:   end if
7: end for
8: return  $Vor_e$ 

```

Algorithmus 5.9 konvertiert Eingabeparameter einer Operation in Vorbedingungen. Für jeden Eingabeparameter der übergebenen Operation (Zeile 2) mit Eigenschaftsreferenz (Zeile 4) wird deren Topologieelement und Eigenschaftenelement des DMMN-Modells in die Menge der Vorbedingungen übernommen.

Algorithmus 5.10 konvertiert Eigenschaftsreferenzen der Ausgabeparameter. Der Algorithmus iteriert über alle Ausgabeparameter einer Operation o (Zeile 3) und untersucht für jede referenzierte Eigenschaft (Zeile 5), ob die Aktion bereits einen Effekt bezüglich dieses Eigenschaftselements spezifiziert (Zeile 9). Existiert für die referenzierte Eigenschaft in Effekte_e schon ein Effekt gleichen Namens (zweite Komponente beider Tupel), wird dieser entfernt (Zeile 10). Dies ist dann der Fall, wenn (i) das Management-Planlet die besagte Eigenschaft selbst beeinflusst oder (ii) eine andere, vom Planlet als Vorbedingung definierte, Operation die Eigenschaft in einem Ausgabeparameter referenziert. Anschließend wird die referenzierte Eigenschaft als Effekt hinzugefügt (Zeile 13). Dabei wird das Wildcard-Symbol als Wert des Effekts spezifiziert, da die Aktion lediglich aussagt, dass die Eigenschaft beeinflusst wird, jedoch nicht welcher konkrete Zielwert angenommen wird. Welchen Wert das Planlet letztendlich für die Eigenschaft setzt, ist somit abhängig von der jeweiligen Annotation bzw. Operation und kann nicht während der Konvertierung spezifiziert werden.

Algorithmus 5.10 KonvertiereAusgaben($o, \text{Effekte}_e$)

```

1: // Erzeuge für alle Ausgabeparameter der Operation, welche Eigenschaftsreferenzen
2: // spezifizieren, entsprechende Effekte
3: for all ( $param_x \in \pi_4(o)$ ) do
4:    $er := \pi_2(param_x)$ 
5:   if ( $er \neq \perp$ ) then
6:      $te_{er} := \pi_1(er)$ 
7:      $e_{er} := \pi_2(er)$ 
8:     // Falls die Aktion die Eigenschaft bereits als Effekt enthält, lösche diesen
9:     if ( $\exists (te_m, ef_n) \in \text{Effekte}_e : te_m = te_{er} \wedge \pi_2(ef_n) = \pi_2(e_{er})$ ) then
10:        $\text{Effekte}_e := \text{Effekte}_e \setminus \{(te_m, ef_n)\}$ 
11:     end if
12:     // Erzeuge zugehöriges Effektelement aus der Eigenschaftsreferenz  $er$ 
13:      $\text{Effekte}_e := \text{Effekte}_e \cup \{(te_{er}, (\pi_1(e_{er}), \pi_2(e_{er}), *, \pi_4(e_{er})))\}$ 
14:   end if
15: end for
16: return  $\text{Effekte}_e$ 

```

5.4.7 Algorithmus zur Berechnung einer partiellen Ordnung von Aktionen

Durch die Konvertierung in Aktionen werden die verschiedenen Klassen an Vorbedingungen und Effekten von Planlets in ein homogenes Datenmodell überführt und die unterschiedlichen Kompatibilitätsdefinitionen durch Bindung an konkrete Topologieelemente des DMMN-Modells aggregiert. In diesem Abschnitt wird ein Algorithmus vorgestellt, der als Subroutine von [Algorithmus 5.1](#) (i) Aktionen aus $\mathfrak{A}_d^{\tilde{\circ}}$ zur Ausführung aller in einem DMMN-Modell deklarierten Managementannotationen selektiert und (ii) eine partielle Ordnung dieser Aktionen generiert, welche die größtmögliche Parallelität bezüglich deren Ausführung erlaubt. Dabei wird garantiert, dass während der Ausführung der Aktionen keine Konflikte zwischen diesen auftreten. Der Algorithmus basiert auf dem *POP-Algorithmus* (engl. „*Partial-Order Planning Algorithm*“) [Wel94] und adaptiert diesen für das DMMN-Metamodell und Planlets ([Abschnitt 5.4.6](#)).

[Algorithmus 5.11](#) beschreibt die durchzuführenden Schritte in Pseudocode. Das Ziel des Algorithmus ist es, eine Menge und Reihenfolge an Aktionen aus $\mathfrak{A}_d^{\tilde{\circ}}$ zu finden, welche den aktuellen Zustand der Anwendung in den Endzustand überführen, der durch die erfolgreiche Abarbeitung aller Managementannotationen in d definiert ist. Um Start- und Endzustand zu beschreiben, werden beide jeweils in Form einer Aktion dargestellt, die im Falle des Startzustands ausschließlich Effekte hat, welche den Ausgangszustand der Anwendung beschreiben, und im Falle des Endzustands als Vorbedingung alle Managementannotationen des DMMN-Modells. Dazu wird über alle Topologieelemente des DMMN-Modells iteriert und die Eigenschaften- sowie Annotationenelemente gesammelt (Zeilen 4–11). Die gesammelten Mengen werden anschließend dazu verwendet, (i) eine künstliche Startaktion α_{Start} (Zeile 15) und (ii) eine künstliche Endaktion α_{Ende} (Zeile 16) zu erzeugen, welche beide in die Menge der verfügbaren statisch kompatiblen Aktionen aufgenommen werden (Zeile 17).

Die Generierung einer partiellen Ordnung von Aktionen, welche den Ausgangszustand in den Zielzustand überführt, basiert auf der Abarbeitung einer dynamisch wachsenden und schrumpfenden Liste aller zu erfüllenden Ziele, die als *Agenda* bezeichnet wird. Sind alle Ziele der Agenda erfüllt, ist die Berechnung

Algorithmus 5.11 GenerierePartielleOrdnung($d, \mathfrak{A}_d^{\approx_\circ}$)

```
1: agenda, zustand, ziele :=  $\emptyset$ 
2: // Die Eigenschaften der Topologieelemente des DMMN-Modells repräsentieren den
3: // Ausgangszustand, deren Managementannotationen das Ziel
4: for all ( $te_i \in TE_d$ ) do
5:   for all ( $e_j \in eigenschaften(te_i)$ ) do
6:     zustand := zustand  $\cup \{(te_i, e_j)\}$ 
7:   end for
8:   for all ( $ma_k \in annotationen_d(te_i)$ ) do
9:     ziele := ziele  $\cup \{(te_i, ma_k)\}$ 
10:  end for
11: end for
12: // Erzeuge künstliche Aktionen zur Spezifikation des Ausgangszustandes und der Ziele.
13: // Da Eigenschaftenelemente und Effektelemente syntaktisch identisch aufgebaut sind,
14: // können die Eigenschaftenelemente der Variable zustand als Effekte behandelt werden.
15:  $\alpha_{Start} := (\emptyset, zustand, \emptyset, \emptyset, \emptyset)$ 
16:  $\alpha_{Ende} := (ziele, \emptyset, \emptyset, \emptyset, \emptyset)$ 
17:  $\mathfrak{A}_d^{\approx_\circ} := \mathfrak{A}_d^{\approx_\circ} \cup \{\alpha_{Start}, \alpha_{Ende}\}$ 
18: // Füge alle Vorbedingungen der künstlichen Endaktion zur Agenda hinzu
19: for all ( $(te_i, ma_j) \in \pi_1(\alpha_{Ende})$ ) do
20:   agenda := agenda  $\cup \{((te_i, ma_j), \alpha_{Ende})\}$ 
21: end for
22: // Erfüllung aller Ziele in der Agenda durch den rekursiven Algorithmus ErfülleZiele
23:  $(\mathcal{A}, \mathcal{O}) := \text{ErfülleZiele}(\{\alpha_{Start}, \alpha_{Ende}\}, \emptyset, \emptyset, agenda, d, \mathfrak{A}_d^{\approx_\circ})$ 
24: // Finale Behebung übrig gebliebener Schreibkonflikte der Aktionen in  $\mathcal{A}$ , welche
25: // nicht durch die bisherige partielle Ordnung gelöst werden
26:  $\mathcal{O} := \text{BehebeSchreibkonflikte}(\mathcal{A}, \mathcal{O})$ 
27: // Rückgabe der selektierten Aktionen und deren partielle Ordnung
28: return  $(\mathcal{A}, \mathcal{O})$ 
```

abgeschlossen. Die Agenda beinhaltet zu Beginn alle Managementannotationen des DMMN-Modells, die als Vorbedingungen des Endzustands α_{Ende} erfasst wurden. Daher wird über alle Vorbedingungen von α_{Ende} iteriert und diese in die Agenda aufgenommen (Zeilen 19–21). Zur Erfüllung der Ziele wird in Zeile 23

die Subroutine *ErfülleZiele* aufgerufen (siehe [Abschnitt 5.4.7.2](#)), welche die Start- und Endzustände in Form der jeweiligen Aktionen, die aktuelle Agenda, sowie das DMMN-Modell und die zu d statisch kompatiblen Aktionen als Eingabe erhält. Basierend auf diesen Eingaben berechnet die Routine (i) eine Menge an Aktionen $\mathcal{A} \subseteq \mathcal{A}_d^{\approx}$ und (ii) eine partielle Ordnung $\mathcal{O}$ über diese Aktionen, deren Ausführung alle in der initialen Agenda enthaltenen Ziele, d. h. alle Annotationen des DMMN-Modells, erreicht. Tritt während der Berechnung ein Fehler auf, z. B. wenn eine Vorbedingung einer Aktion nicht erfüllt werden kann, existiert keine partielle Ordnung an Aktionen, um alle Annotationen in d auszuführen, d. h. es kann kein Managementplan generiert werden.

Management-Planlets kommunizieren Informationen untereinander über die Eigenschaften und Zustände der Anwendungskomponenten sowie deren Relationen zueinander, welche in Form des Instanzmodells der Anwendung strukturiert erfasst sind. Beeinflussen mehrere Planlets dieselbe Eigenschaft einer Komponente oder Relation, dürfen diese nicht parallel ausgeführt werden, da sich die Planlets andernfalls bei ihrer Ausführung semantisch stören können und das korrekte Endresultat beider Ausführungen nicht gewährleistet werden kann. Beispielsweise, wenn zwei unterschiedliche Management-Planlets Software auf einem Betriebssystem installieren, welche beide eine bestimmte Eigenschaft des Systems beeinflussen. Aufgrund ähnlicher Faktoren dürfen zwei Planlets, von denen eines eine Eigenschaft beeinflusst und ein anderes diese Eigenschaft als Vorbedingung spezifiziert, ebenfalls nicht parallel ausgeführt werden, da eine korrekte Funktionsweise des Planlets mit der Vorbedingung nicht gewährleistet werden kann, wenn diese Eigenschaft gleichzeitig durch das andere Planlet einer Änderung unterzogen wird. Die Subroutine *ErfülleZiele* behandelt die zweite Art der Konflikte durch Ordnung aller in Konflikt stehender Aktionen. Konflikte der ersten Art werden abschließend in Zeile 26 behoben. Diese Gedankengänge beruhen auf der Analyse paralleler Ausführungen von Prozessen durch A. J. Bernstein [[Ber66](#)], welche auch als *Bernstein Kriterium* (engl. „*Bernstein Criterion*“) bekannt sind [[Bae73](#); [LA94](#)]. Das Resultat der Berechnung ist ein Paar bestehend aus (i) den selektierten Aktionen $\mathcal{A}$ und (ii) der partiellen Ordnung $\mathcal{O}$ dieser Aktionen, welche alle Annotationen in d ausführen.

5.4.7.1 Dynamische Kompatibilität zwischen Management-Planlets

Die Routine *ErfülleZiele*, die im nächsten Abschnitt eingeführt wird, berechnet aus der Menge der statisch kompatiblen Aktionen $\mathfrak{A}_d^{\approx_\circ}$ eine Auswahl und partielle Ordnung, welche alle im DMMN-Modell d deklarierten Managementannotationen ausführt. Zum Zeitpunkt der Ausführung einer Aktion, d. h. des dadurch repräsentierten Planlets, müssen jedoch sowohl deren statische als auch dynamische Vorbedingungen erfüllt sein (vgl. [Abschnitt 5.4.2](#)). Während die statischen Vorbedingungen aller Aktionen bereits durch die Menge $\mathfrak{A}_d^{\approx_\circ}$ berücksichtigt sind, müssen die dynamischen Vorbedingungen bezüglich Eigenschaften von Komponenten und Relationen, welche im Topologiefragment des zugehörigen Management-Planlets spezifiziert sind, bei der Auswahl und Ordnung der Aktionen beachtet werden. Da die Effekte von Management-Planlets die Eigenschaften von Komponenten und Relationen beeinflussen, muss die *dynamische Kompatibilität* zwischen Effekten und Eigenschaftsvorbedingungen von Aktionen formal definiert werden. Diese Kompatibilitätsrelation wird im Folgenden zur Berechnung verwendet, welche Aktionen die Ausführung anderer Aktionen ermöglichen, d. h. welche Aktionen andere Aktionen *aktivieren*.

Ein Effektelement einer Aktion ist kompatibel zu einem Eigenschaftenelement einer anderen Aktion, also einer Vorbedingung, wenn die beiden Elemente bezüglich der Attribute „Name“ und „Datentyp“ übereinstimmen. Falls die geforderte Eigenschaft einen konkreten „Wert“ spezifiziert, muss dieser zum Wert des Effekts kompatibel sein. Dies wird folgendermaßen formalisiert:

Definition 70 (Kompatibilität zwischen Effekt- und Eigenschaftenelementen).

Sei $ef \in EF$ ein Effektelement und $e \in E$ ein Eigenschaftenelement, dann wird die heterogene Kompatibilitätsrelation $\approx_{ef} \subseteq EF \times E$ wie folgt definiert:

$$\begin{aligned} ef \approx_{ef} e &: \Leftrightarrow \pi_2(ef) = \pi_2(e) \wedge \\ &\pi_4(ef) = \pi_4(e) \wedge \\ &(\pi_3(ef) = \pi_3(e) \vee (\pi_3(e) = * \wedge \pi_3(ef) \neq \perp) \vee \pi_3(e) = \perp) \end{aligned}$$

■

5.4.7.2 Algorithmus zur Erfüllung von Zielen in der Agenda

Algorithmus 5.12 zeigt den Algorithmus zur Erfüllung aller Ziele. Die Agenda beinhaltet Tupel $((te_g, x_g), a_{Ziel})$, deren Komponenten te_g ein Topologieelement aus d darstellt, x_g ein Eigenschaften- bzw. Annotationenelement von te_g und a_{Ziel} eine Aktion, welche diese als Vorbedingung spezifiziert. Beim ersten Aufruf der Routine enthält die Agenda alle Vorbedingungen des Endzustandes a_{Ende} , d. h. alle Annotationen der Topologieelemente des DMMN-Modells d (vgl. Zeilen 19–21 in **Algorithmus 5.11**). Die Menge $\mathcal{A}$ enthält alle zum Zeitpunkt des Aufrufs bereits ausgewählten Aktionen, $\mathcal{O}$ alle bisher festgelegten Ordnungen zwischen Aktionen in $\mathcal{A}$. Ist die Agenda leer, d. h. erfüllen $\mathcal{A}$ und $\mathcal{O}$ alle Ziele, werden die beiden Mengen als Ergebnis zurückgeliefert (Zeilen 2–4). Falls die Agenda noch unerfüllte Ziele beinhaltet, wählt die Routine bei jedem Aufruf nicht-deterministisch (i) ein Ziel aus der Agenda (Zeile 10) sowie (ii) eine Aktion a_{Neu} aus $\mathfrak{A}_d^{\approx}$, welche dieses Ziel erfüllt (Zeilen 12–15). Ein Ziel ist entweder eine Annotation des Endzustandes a_{Ende} , oder eine Vorbedingung einer bereits ausgewählten Aktion. Die Aktion a_{Neu} wird dabei so gewählt, dass sie konsistent vor a_{Ziel} geordnet werden kann (Zeile 12) und keine Annotationen ausführt, die bereits von anderen Aktionen in $\mathcal{A}$ ausgeführt werden (Zeile 13). Existiert keine solche Aktion a_{Neu} , bricht der Algorithmus mit einem Fehler ab.

Ist die Aktion a_{Neu} noch nicht in $\mathcal{A}$ enthalten, wird sie hinzugefügt (Zeile 18) und spezifiziert, dass a_{Neu} nach Startzustand a_{Start} und vor Endzustand a_{Ende} ausgeführt werden muss (Zeile 19). Zudem werden alle Vorbedingungen der gewählten Aktion a_{Neu} in die Agenda aufgenommen, da diese noch erfüllt werden müssen, um das von a_{Neu} repräsentierte Planlet auszuführen (Zeilen 21–23). Um festzuhalten, welches Ziel a_{Neu} erfüllt, werden in der Menge $\mathcal{L}$ alle *kausalen Links* erfasst (engl. „causal link“ [Wel94]). Ein kausaler Link $a_{Neu} \xrightarrow{te_g, x_g} a_{Ziel}$ dokumentiert, dass a_{Neu} die Vorbedingung (te_g, x_g) von a_{Ziel} erfüllt (Zeile 26). Diese Information wird dazu verwendet, um beim Hinzufügen weiterer Aktionen zu überprüfen, ob diese bereits bestehende Abhängigkeiten zwischen Aktionen verletzen (engl. „threat“ [Wel94]). Zudem wird die Ordnung festgehalten, dass a_{Neu} vor a_{Ziel} ausgeführt werden muss (Zeile 27). Da a_{Neu} das Ziel $((te_g, x_g), a_{Ziel})$ erfüllt, wird dieses aus der Agenda gelöscht (Zeile 28).

Algorithmus 5.12 ErfülleZiele($\mathcal{A}, \mathcal{O}, \mathcal{L}, \text{agenda}, d, \mathfrak{A}_d^{\approx_d^\circ}$)

```

1: // Wenn alle Vorbedingungen erfüllt sind, terminiere
2: if ( $\text{agenda} = \emptyset$ ) then
3:   return ( $\mathcal{A}, \mathcal{O}$ )
4: end if

5: // Erstelle lokale Kopien
6:  $\mathcal{A}' := \mathcal{A}$ 
7:  $\mathcal{O}' := \mathcal{O}$ 
8:  $\text{agenda}' := \text{agenda}$ 

9: // Wähle ein unerfülltes Ziel aus der Agenda (Nicht-Determinismus)
10: choose  $((te_g, x_g), a_{Ziel}) \in \text{agenda}$ 

11: // Wähle eine Aktion, welche das gewählte Ziel erfüllt (Nicht-Determinismus)
12: choose  $a_{Neu} \in \{a \in \mathfrak{A}_d^{\approx_d^\circ} \mid (a \neq a_{Ziel} \wedge (a_{Ziel}, a) \notin \mathcal{O}'^+) \wedge$ 
13:    $(a \notin \mathcal{A} \Rightarrow \forall a_i \in \mathcal{A} : \pi_3(a) \cap \pi_3(a_i) = \emptyset) \wedge$ 
14:    $(\exists (te_a, ef_a) \in \pi_2(a) : te_a = te_g \wedge ef_a \approx_{ef} x_g) \vee$ 
15:    $(\exists (te_a, ma_a) \in \pi_3(a) : te_a = te_g \wedge ma_a = x_g))\}$ 

16: // Gewählte Aktion aufnehmen falls noch nicht in  $\mathcal{A}'$  enthalten
17: if ( $a_{Neu} \notin \mathcal{A}'$ ) then
18:    $\mathcal{A}' := \mathcal{A}' \cup \{a_{Neu}\}$ 
19:    $\mathcal{O}' := \mathcal{O}' \cup \{a_{Start} < a_{Neu}\} \cup \{a_{Neu} < a_{Ende}\}$ 
20:   // Füge alle zustandsbezogenen Vorbedingungen der Aktion in die Agenda hinzu
21:   for all  $((te_i, e_j) \in \pi_1(a_{Neu}))$  do
22:      $\text{agenda}' := \text{agenda}' \cup \{((te_i, e_j), a_{Neu})\}$ 
23:   end for
24: end if

25: // Kausalen Link erstellen, partielle Ordnung definieren und Agenda aktualisieren
26:  $\mathcal{L}' := \mathcal{L} \cup \{a_{Neu} \xrightarrow{te_g, x_g} a_{Ziel}\}$ 
27:  $\mathcal{O}' := \mathcal{O}' \cup \{a_{Neu} < a_{Ziel}\}$ 
28:  $\text{agenda}' := \text{agenda}' \setminus \{((te_g, x_g), a_{Ziel})\}$ 

29: // Reihenfolgespezifikationen in partielle Ordnung konvertieren
30:  $\mathcal{O}' := \mathcal{O}' \cup \{\text{KonvertiereReihenfolgespezifikationen}(a_{Neu}, \mathcal{A}', \mathcal{O}', d)\}$ 

31: // Löse Konflikte kausaler Links und von Schreibvorgängen
32:  $\mathcal{O}' := \mathcal{O}' \cup \{\text{BehebeKausaleKonflikte}(\mathcal{A}', \mathcal{O}', \mathcal{L}', d)\}$ 
33: return ErfülleZiele( $\mathcal{A}', \mathcal{O}', \mathcal{L}', \text{agenda}', d, \mathfrak{A}_d^{\approx_d^\circ}$ )

```

Spezifiziert das DMMN-Modell Ausführungsreihenfolgen bezüglich Managementannotationen, müssen diese bei der Generierung der partiellen Ordnung beachtet werden, indem die Aktionen, welche Annotationen ausführen für die Reihenfolgespezifikationen im DMMN-Modell festgelegt sind, entsprechend geordnet werden. Dies wird durch die Subroutine *KonvertiereReihenfolgespezifikationen* in Zeile 30 durchgeführt, die in [Abschnitt 5.4.7.3](#) beschrieben wird.

Das Einfügen von α_{Neu} steht eventuell mit anderen Aktionen, die bereits in $\mathcal{A}$ enthalten sind, in Konflikt: Beeinflussen mehrere Aktionen dieselbe Eigenschaft einer Komponente oder Relation, welche Vorbedingung einer anderen Aktion ist, dürfen diese Aktionen nicht parallel ausgeführt werden und müssen geordnet werden. Zur Behebung dieser Konflikte wird in Zeile 32 die Subroutine *BehebeKausaleKonflikte* aufgerufen, die in [Abschnitt 5.4.7.4](#) beschrieben wird.

5.4.7.3 Algorithmus zur Konvertierung von Reihenfolgespezifikationen

In diesem Abschnitt wird beschrieben, wie Reihenfolgespezifikationen, welche die Ausführungsreihenfolge von Managementannotationen in DMMN-Modellen festlegen, die partielle Ordnung von Aktionen bei der Plangenerierung beeinflussen und wie diese in die zweistellige Ordnungsrelation $\mathcal{O}$ über Aktionen konvertiert werden. [Algorithmus 5.13](#) zeigt die Schritte zur Konvertierung in Pseudocode. Für jede Managementannotation ma_{t_1} , welche die neu eingefügte Aktion α_{Neu} in d ausführt (Zeile 5), wird überprüft, ob bereits eine andere Aktionen α_i in $\mathcal{A}$ enthalten ist, welche eine Managementannotation ma_{t_2} in d ausführt (Zeile 9), für die bezüglich der von α_{Neu} ausgeführten Annotation ma_{t_1} eine Ausführungsreihenfolge spezifiziert ist (Zeile 12 bzw. Zeile 15). Ist dies der Fall, werden die Aktionen α_{Neu} und α_i gemäß der Reihenfolge von ma_{t_1} und ma_{t_2} geordnet (Zeile 13 bzw. Zeile 16). Möglicherweise steht diese Reihenfolge mit bereits existierenden kausalen Abhängigkeiten in Konflikt, wodurch die Konsistenz von $\mathcal{O}'$ verletzt wird (Zeile 20). In diesem Fall kann aufgrund der per definitionem korrekten nicht-deterministischen Entscheidungen keine konsistente partielle Ordnung generiert werden, wodurch die Plangenerierung abgebrochen werden muss. Ist die Ordnung konfliktfrei, wird $\mathcal{O}'$ als Ergebnis zurückgegeben. Die Routine wird von [Algorithmus 5.12](#) in Zeile 30 aufgerufen.

Algorithmus 5.13 KonvertiereReihenfolgespezifikationen($\alpha_{\text{Neu}}, \mathcal{A}, \mathcal{O}, d$)

```
1: // Erstelle lokale Kopie von  $\mathcal{O}$ 
2:  $\mathcal{O}' := \mathcal{O}$ 
3: // Überprüfe für alle von  $\alpha_{\text{Neu}}$  ausgeführten Annotationen, ob in  $d$  Reihenfolgen bezüglich
4: // lich Annotationen spezifiziert sind, die von anderen Aktionen ausgeführt werden
5: for all  $((ma_{pl1}, ma_{t1}) \in \pi_2(\pi_5(\alpha_{\text{Neu}})))$  do
6:     // Beziehe dazu alle Aktionen mit ein, die bereits in  $\mathcal{A}$  enthalten sind
7:     for all  $(\alpha_i \in \mathcal{A})$  do
8:         // Überprüfe alle Managementannotationen  $ma_{t2}$ , die  $\alpha_i$  ausführt
9:         for all  $((ma_{pl2}, ma_{t2}) \in \pi_2(\pi_5(\alpha_i)))$  do
10:             // Falls in  $d$  eine Ausführungsreihenfolge für  $ma_{t1}$  und  $ma_{t2}$  modelliert ist,
11:             // übertrage diese als partielle Ordnung auf die zugehörigen Aktionen
12:             if  $((ma_{t1}, ma_{t2}, O) \in P_d)$  then
13:                  $\mathcal{O}' := \mathcal{O}' \cup \{\alpha_{\text{Neu}} < \alpha_i\}$ 
14:             end if
15:             if  $((ma_{t2}, ma_{t1}, O) \in P_d)$  then
16:                  $\mathcal{O}' := \mathcal{O}' \cup \{\alpha_i < \alpha_{\text{Neu}}\}$ 
17:             end if
18:             // Falls für zwei Aktionen eine widersprüchliche Ausführungsreihenfolge
19:             // spezifiziert ist, d. h.  $\mathcal{O}'$  ist nicht konsistent, werfe Fehler
20:             if  $(\exists \alpha_1, \alpha_2 \in \mathcal{A} : \{\alpha_1 < \alpha_2\} \in \mathcal{O}'^+ \wedge \{\alpha_2 < \alpha_1\} \in \mathcal{O}'^+)$  then
21:                 // Werfe Fehler und breche die Berechnung ab: Es kann kein
22:                 // Managementplan generiert werden
23:                 throw fault
24:             end if
25:         end for
26:     end for
27: end for
28: return  $\mathcal{O}'$ 
```

5.4.7.4 Algorithmus zur Behebung kausaler Konflikte

Das Einfügen von Aktionen kann zu kausalen Konflikten mit anderen Aktionen führen, die bereits in $\mathcal{A}$ enthalten sind. [Algorithmus 5.14](#) beschreibt, wie diese Konflikte durch Ordnung der in Konflikt stehenden Aktionen gelöst werden können. Ein kausaler Konflikt tritt immer dann auf, wenn [Algorithmus 5.12](#) eine neue Aktion a_{Neu} in $\mathcal{A}$ eingefügt hat (Zeile 18), welche als Effekt eine Eigenschaft (te_g, e_g) beeinflusst, für die ein kausaler Link $a_1 \xrightarrow{te_g, e_g} a_2$ zwischen zwei anderen Aktionen a_1 und a_2 in $\mathcal{L}$ existiert. Wird a_{Neu} parallel zu a_1 und a_2 ausgeführt, besteht die Gefahr, dass a_1 während seiner Ausführung durch die Ausführung von a_{Neu} gestört wird bzw. die Vorbedingung von a_2 , d. h. (te_g, e_g) , durch die Ausführung von a_{Neu} nicht mehr erfüllt ist. Daher müssen in Konflikt stehende Aktionen geordnet werden, sodass a_{Neu} entweder vor oder nach a_1 und a_2 ausgeführt wird, aber nicht parallel (*Isolationsprinzip*).

Dazu überprüft [Algorithmus 5.14](#) für jede Aktion a_i in $\mathcal{A}$ (Zeile 4), ob deren Effekte (Zeile 5) mit kausalen Links zwischen anderen Aktionen a_p und a_q in $\mathcal{A}$ in Konflikt stehen (Zeilen 8–11). Die aktuelle Ordnungsrelation $\mathcal{O}'$, welche Anfangs in Zeile 2 aus $\mathcal{O}$ kopiert wurde, spezifiziert möglicherweise bereits implizit oder explizit, dass a_i vor bzw. nach den kausal verknüpften Aktionen a_p und a_q ausgeführt wird und der Konflikt somit bereits durch die aktuelle Ordnung $\mathcal{O}'$ gelöst ist. Daher wird untersucht, ob die transitive Hülle $\mathcal{O}'^+$ bereits eine konfliktlösende Ordnung enthält (Zeile 14). Falls die transitive Hülle $\mathcal{O}'^+$ keine konfliktlösende Ordnung enthält, wird nicht-deterministisch eine konfliktlösende Ordnung hinzugefügt (Zeile 16), indem a_i entweder vor oder nach a_q geordnet wird. Gibt es keine Konfliktlösung, wird ein Fehler geworfen und die Berechnung dadurch abgebrochen (Zeile 20). Aufgrund der per definitionem korrekten nicht-deterministischen Entscheidungen der in diesem Kapitel vorgestellten Algorithmen existiert im Falle eines geworfenen Fehlers keine Menge an Aktionen, die partiell so geordnet werden kann, dass ein vollständiger und konfliktfreier Managementplan generierbar ist.

Algorithmus 5.14 BehebeKausaleKonflikte($\mathcal{A}, \mathcal{O}, \mathcal{L}, d$)

```
1: // Erstelle lokale Kopie von  $\mathcal{O}$ 
2:  $\mathcal{O}' := \mathcal{O}$ 
3: // Iteriere über alle Effekte aller Aktionen in  $\mathcal{A}$ , die Eigenschaften betreffen
4: for all ( $\alpha_i \in \mathcal{A}$ ) do
5:   for all ( $(te, ef) \in \pi_2(\alpha_i)$ ) do
6:     // Überprüfe für alle kausalen Links zwischen zwei Aktionen  $\alpha_p$  und  $\alpha_q$ , mit
7:     // denen  $\alpha_i$  in Konflikt steht, ob eine Ordnung erforderlich ist
8:     for all ( $\alpha_p \xrightarrow{te_g, x_g} \alpha_q \in \mathcal{L} \mid x_g \in E \wedge te = te_g \wedge \pi_2(ef) = \pi_2(x_g)$ ) do
9:       // Nur kausale Links untersuchen, an denen  $\alpha_i$  nicht beteiligt ist, da eine
10:      // Aktion nicht mit sich selbst in Konflikt stehen kann
11:      if ( $\alpha_i \neq \alpha_p \wedge \alpha_i \neq \alpha_q$ ) then
12:        // Ordnung zwischen  $\alpha_i$  und  $\alpha_p$  bzw.  $\alpha_i$  und  $\alpha_q$  nur dann explizit
13:        // festlegen, falls diese nicht aus der transitiven Hülle von  $\mathcal{O}'$  folgt
14:        if ( $\neg((\alpha_i < \alpha_p) \in \mathcal{O}'^+ \vee (\alpha_q < \alpha_i) \in \mathcal{O}'^+)$ ) then
15:          // Ordne  $\alpha_i$  vor  $\alpha_p$  oder nach  $\alpha_q$  (Nicht-Determinismus)
16:           $\mathcal{O}' := \mathcal{O}' \cup \{\alpha_i < \alpha_p\} \vee \mathcal{O}' := \mathcal{O}' \cup \{\alpha_q < \alpha_i\}$ 
17:        end if
18:        // Falls für zwei Aktionen eine widersprüchliche Ausführungsreihen-
19:        // folge spezifiziert ist, d. h.  $\mathcal{O}'$  ist nicht konsistent, werfe Fehler
20:        if ( $\exists \alpha_1, \alpha_2 \in \mathcal{A} : \{\alpha_1 < \alpha_2\} \in \mathcal{O}'^+ \wedge \{\alpha_2 < \alpha_1\} \in \mathcal{O}'^+$ ) then
21:          // Werfe Fehler und breche die Berechnung ab: Es kann kein
22:          // Managementplan generiert werden
23:          throw fault
24:        end if
25:      end if
26:    end for
27:  end for
28: end for
29: return  $\mathcal{O}'$ 
```

5.4.7.5 Algorithmus zur Behebung von Schreibkonflikten

Nach der Erfüllung aller Ziele in Zeile 23 von [Algorithmus 5.11](#) existiert eine Menge $\mathcal{A}$ an Aktionen sowie eine partielle Ordnung $\mathcal{O}$, die alle Annotationen im DMN-Modell ausführen. Möglicherweise sind jedoch noch Aktionen in $\mathcal{A}$ enthalten, welche dieselben Eigenschaften beeinflussen, jedoch nicht mit kausalen Links in Konflikt stehen. Diese Aktionen dürfen ebenfalls nicht parallel zueinander ausgeführt werden, da sie sich sonst gegenseitig stören können.

Algorithmus 5.15 BehebeSchreibkonflikte($\mathcal{A}, \mathcal{O}$)

```
1: // Erstelle lokale Kopie von  $\mathcal{O}$ 
2:  $\mathcal{O}' := \mathcal{O}$ 

3: // Spezifiziere Ordnung für alle Aktionen, welche dieselben Eigenschaften be-
4: // einflussen und nicht implizit oder explizit durch  $\mathcal{O}'^+$  geordnet sind
5: for all ( $\alpha_i, \alpha_j \in \mathcal{A} \mid \alpha_i \neq \alpha_j$ ) do
6:     // Extrahiere die Mengen der Effekte auf Eigenschaften beider Aktionen
7:      $Ef_i := \pi_2(\alpha_i)$ 
8:      $Ef_j := \pi_2(\alpha_j)$ 
9:     // Überprüfe, ob die beiden Aktionen  $\alpha_i$  und  $\alpha_j$  dieselben Eigenschaften beeinflussen
10:    if ( $\exists (te_1, ef_1) \in Ef_i \exists (te_2, ef_2) \in Ef_j \mid te_1 = te_2 \wedge \pi_2(ef_1) = \pi_2(ef_2)$ ) then
11:        // Falls  $\alpha_i$  und  $\alpha_j$  nicht implizit oder explizit über  $\mathcal{O}'$  geordnet sind, lege
12:        // eine Ordnung für die Aktionen fest (Nicht-Determinismus)
13:        if ( $\neg((\alpha_i < \alpha_j) \in \mathcal{O}'^+ \vee (\alpha_j < \alpha_i) \in \mathcal{O}'^+)$ ) then
14:             $\mathcal{O}' := \mathcal{O}' \cup \{\alpha_i < \alpha_j\} \vee \mathcal{O}' := \mathcal{O}' \cup \{\alpha_j < \alpha_i\}$ 
15:        end if
16:    end if
17: end for
18: return  $\mathcal{O}'$ 
```

[Algorithmus 5.15](#) überprüft daher paarweise alle Aktionen in $\mathcal{A}$ (Zeile 5), ob sie dieselbe Eigenschaft beeinflussen (Zeile 10) und nicht implizit oder explizit durch $\mathcal{O}'$ geordnet sind, wozu deren Mitgliedschaft in der transitiven Hülle $\mathcal{O}'^+$ von $\mathcal{O}'$ untersucht wird (Zeile 13). Falls keine Ordnung spezifiziert ist, wird diese nicht-deterministisch festgelegt, um den Konflikt zu lösen (Zeile 14).

5.5 Transformation in ein imperatives Managementmodell

In den vorangegangenen Abschnitten wurden generische, technologieunabhängige Algorithmen zur Generierung partieller Ordnungen von Planlets vorgestellt, welche zur Ausführung in imperative Managementmodelle übersetzt werden müssen. Da sich die Workflow-Technologie aufgrund der in [Abschnitt 3.2](#) erhobenen Anforderungen gegenwärtig am besten zur Umsetzung imperativer Managementmodelle eignet, wird in [Abschnitt 5.5.1](#) zunächst eine Realisierung unabhängig von konkreten Workflow-Sprachen beschrieben. Anschließend wird das Konzept in [Abschnitt 5.5.2](#) auf den Standard BPEL angewendet.

5.5.1 Generierung imperativer, graphbasierter Managementpläne

Die Transformation der Mengen $\mathcal{A}$ und $\mathcal{O}$ in ein imperatives Managementmodell auf Basis der Workflow-Technologie entspricht der *Generierung eines Managementplans* (vgl. [Abschnitt 5.2.2](#)) und implementiert die in [Zeile 11](#) in [Algorithmus 5.1](#) aufgerufene Routine *GeneriereImperativesManagementmodell*($\mathcal{A}, \mathcal{O}$). Management-Planlets und Managementpläne werden mittels imperativer, graphbasierter Workflow-Sprachen [[KMWL09](#)] realisiert, welche mindestens folgende Workflow-Muster [[vAtHKB03](#)] unterstützen: (i) *Sequenzmuster* (engl. „Sequence Pattern“), d. h. Kontrollflusskanten zwischen Aktivitäten ohne Übergangsbedingungen (engl. „transition conditions“), (ii) das *Parallel Split Pattern*, das die parallele Ausführung von Aktivitäten ermöglicht, sowie (iii) das *Synchronisationsmuster* (engl. „Synchronization Pattern“), das die Zusammenführung paralleler Ausführungen ermöglicht und diese synchronisiert. Management-Planlets werden als traditionelle Subprozesse [[KELU10](#)] modelliert, die somit dem SESE-Schema folgen und genau einen Anfang und ein Ende haben (vgl. [Abschnitt 5.2.1](#)). Planlets werden *hierarchisch* als *Kindprozesse* von einem *Vaterprozess* [[LR00](#)], dem generierten Managementplan, aufgerufen und übermitteln an diesen nach Ausführung das Ergebnis, d. h. ob die vom Planlet auszuführenden Managementaufgaben erfolgreich durchgeführt werden konnten oder ob nicht behandelbare Fehler aufgetreten sind. Weitere Interaktionen mit dem Vaterprozess sind während der Ausführung eines Planlets nicht möglich.

Die Modellierung von Planlets als hierarchische Subprozesse sowie deren Aufrufe durch den Managementplan kann auf zwei Arten durchgeführt werden. Zum einen können die Subprozesse direkt in den generierten Managementplan eingebettet werden (engl. „*Inline Sub-process*“ [KKL+05]), wodurch diese Teil des Plans werden. Zum anderen können Planlets auch als eigenständige, verteilte Prozesse realisiert werden (engl. „*Standalone Sub-process*“ [KKL+05]), die vom Managementplan aufgerufen werden aber selbst nicht im Plan enthalten sind (engl. „*Orchestration*“). Folgende Aspekte sind bezüglich der PALMA-Methode entscheidend: Die Einbettung von Planlets resultiert in Managementplänen, welche die gesamte Managementlogik enthalten. Diese kann somit in ihrer Gesamtheit anhand eines einzelnen Prozessmodells adaptiert werden (engl. „*self-containment*“). Das Auslagern der Subprozesse ermöglicht hingegen eine effiziente Skalierung durch Ausführungsumgebungen, erschwert jedoch die Modifizierung der Managementlogik, da diese bezüglich Änderungswünschen exklusiv und individuell bereitgestellt werden müssen. Die direkte Adaption der Managementfunktionalität ausschließlich anhand des generierten Prozessmodells ist somit nicht möglich. Dies ist jedoch eine grundlegende Anforderung der PALMA-Methode (vgl. [Anforderung 5](#)), weshalb die direkte Einbettung gewählt wird. Die Auslagerung von Planlets als eigenständige, entfernte Prozesse (engl. „*Remote Sub-process*“ [LR00]) wird in [Abschnitt 5.6.6](#) im Rahmen zukünftiger Arbeiten diskutiert. Unabhängig von deren Umsetzung geben Planlets ihre Autonomie bezüglich der Lebensdauer zugunsten des Managementplans auf, was der Definition eines traditionellen Subprozesses entspricht [KELU10].

Die Aufrufe der ausgewählten Planlets, welche in $\mathcal{A}$ jeweils als Aktion inklusive zugehöriger statisch kompatibler Zuordnung enthalten sind, werden durch Aktivitäten realisiert, welche die jeweiligen Planlets als Subprozesse einbetten und die benötigten Eingaben, unter anderem die Zuordnung und das zugehörige DMMN-Modell, übermitteln (siehe [Abschnitt 5.2.3](#)). Die partielle Ordnung $\mathcal{O}$ wird in Kontrollflusskanten übersetzt, welche die entsprechenden Workflow-Aktivitäten gemäß der von $\mathcal{O}$ spezifizierten Ausführungsreihenfolge miteinander verbinden. Management-Planlets, welche nicht durch $\mathcal{O}$ geordnet sind, werden somit parallel ausgeführt. Dieses Konzept wird in [Abbildung 5.12](#) skizziert.

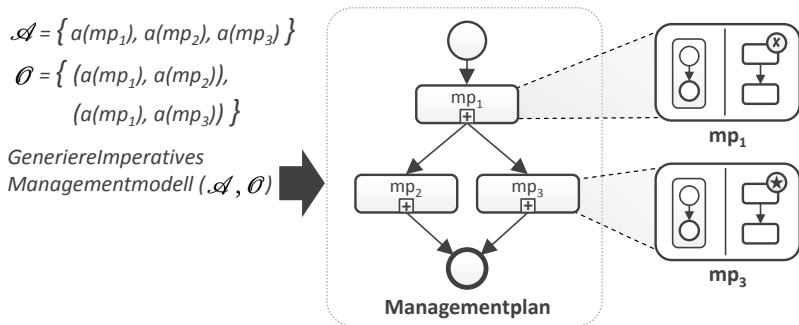


Abbildung 5.12: Generierung eines imperativen Plans aus $\mathcal{O}$ und $\mathcal{A}$

Der generierte Managementplan sichert zu, dass entweder alle enthaltenen Planlets erfolgreich ausgeführt werden oder die Ausführung im Falle eines Fehlers abgebrochen und alle bisher getätigten Änderungen rückgängig gemacht werden. Die eingebetteten Planlets müssen dazu die in [Abschnitt 5.2.4](#) beschriebenen Mechanismen umsetzen: Tritt während der Ausführung ein nicht behandelbarer Fehler auf, muss der jeweilige Subprozess, d. h. das Planlet, alle getätigten Änderungen selbständig rückgängig machen und den Vaterprozess über die gescheiterte Ausführung informieren. Dieser terminiert alle noch laufenden Subprozesse und kompensiert alle zu diesem Zeitpunkt erfolgreich ausgeführten, abgeschlossenen Subprozesse in umgekehrter Ausführungsreihenfolge, was dem Ansatz der *Compensation Spheres* [[Ley95](#); [LR00](#)] entspricht¹. Dazu müssen Planlets folgende Mechanismen implementieren: Wird die Kompensation des Subprozesses vom Vaterprozess, d. h. dem Managementplan, angestoßen, muss der Subprozess seine erfolgreich abgeschlossene Ausführung vollständig rückgängig machen. Wird die Terminierung des Subprozesses angestoßen, muss dieser seine aktuelle Ausführung abbrechen und alle bisher getätigten Änderungen rückgängig machen. Dies bedarf der Implementierung expliziter Geschäftslogik, etwa durch Planlet-spezifische „*Compensation*–“ und „*Termination Handler*“ [[OAS07b](#)]. In allen Fällen aktualisiert der Subprozess das Instanzmodell der Anwendung gemäß aller getätigten Rücksetzungen.

¹Eine Erweiterung des SAGA-Ansatzes [[GS87](#)] auf nicht-lineare Prozesse

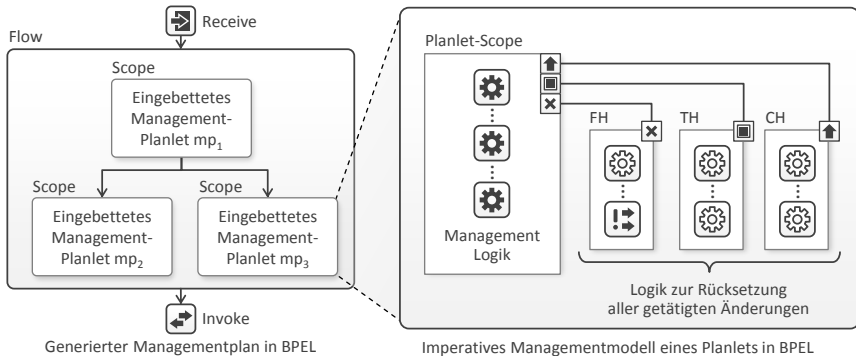


Abbildung 5.13: Technische Umsetzung von Managementplan (links) und Management-Planlet (rechts) in standardkonformem BPEL

5.5.2 Realisierung von Managementplänen und Planlets in BPEL

In diesem Kapitel wird die Realisierung von Managementplänen und Planlets in der Workflow-Sprache BPEL vorgestellt, welche sich aufgrund der Standardisierung zur Modellierung portabler Prozessmodelle eignet. Dazu wird (i) die Transformation einer partiellen Ordnung $(\mathcal{A}, \mathcal{O})$ in einen BPEL-Workflow definiert und (ii) ein Modellierungsschema für Planlets in BPEL vorgestellt, welches die in [Abschnitt 5.5.1](#) definierten Konzepte unterstützt. Die Realisierung ist exemplarisch und kann ähnlich auf andere imperative, graphbasierte Prozessmodellierungssprachen, wie beispielsweise BPMN, übertragen werden.

Managementpläne werden durch eine eingehende Nachricht gestartet, was in BPEL durch eine `bpel:receive`-Aktivität modelliert wird. Nach Ausführung der Managementaufgaben sendet der Managementplan mittels einer `bpel:invoke`-Aktivität eine Nachricht an den Aufrufenden zurück. Managementpläne in BPEL werden somit asynchron aufgerufen, wozu die Adresse des Aufrufenden (engl. „callback“) und andere Informationen, wie z. B. ein Korrelationsbezeichner, in der eingehenden Nachricht übertragen werden. Die Managementlogik wird innerhalb einer `bpel:flow`-Aktivität ausgeführt, welche die Parallelität der Planlet-Ausführungen ermöglicht. Die genannten Aktivitäten sind für jede Plangenerierung identisch und bilden somit den Rahmen für die individuelle

Managementlogik, welche im Folgenden in Form von Management-Planlets hinzugefügt wird. Diese Struktur wird in [Abbildung 5.13](#)¹ auf der linken Seite skizziert. Wie im vorangegangenen Abschnitt definiert, werden Planlets direkt in den Managementplan eingebettet. Dazu wird für jede Aktion in $\mathcal{A}$ eine strukturelle `bpel:scope`-Aktivität erzeugt, in die das Workflow-Modell des jeweiligen Planlets eingefügt wird. Die partielle Ordnung $\mathcal{O}$, welche die Reihenfolge der Planlets spezifiziert, wird durch Kontrollflusskanten (`bpel:link`) abgebildet, welche die jeweiligen Scopes der Management-Planlets verbinden.

Management-Planlets bestehen in BPEL, wie in [Abbildung 5.13](#) auf der rechten Seite skizziert, aus vier Teilen. (i) Zum einen implementiert ein Planlet die jeweilige Managementlogik eingebettet in einen individuellen *Planlet-Scope*, was in der Abbildung durch ausgefüllte Zahnräder dargestellt wird. Da BPEL vorwiegend zur Orchestrierung externer Dienste eingesetzt wird und dafür die nötigen Mechanismen bereitstellt, wird in [Abschnitt 5.5.3](#) erörtert, wie die Managementlogik technisch durch Planlets in BPEL realisiert werden kann. Tritt bei der Planlet-Ausführung ein Fehler auf, müssen die in [Abschnitt 5.5.1](#) vorgestellten Mechanismen ausgeführt werden. Dazu werden *Handler* eingesetzt, welche mit dem Planlet-Scope assoziiert sind². (ii) Falls während der Ausführung im Planlet-Scope ein Fehler auftritt, sind `bpel:faultHandlers` (FH) für die vollständige Rücksetzung aller getätigten Änderungen und die Weiterleitung des Fehlers an den Vater-Scope durch ein `bpel:rethrow` zuständig. (iii) Gleichermaßen implementiert jedes Planlet einen `bpel:terminationHandler` (TH), der die laufende Ausführung des Planlets unterbricht und alle getätigten Änderungen rückgängig macht. Ist die Ausführung eines Planlets erfolgreich abgeschlossen und es tritt bei einem anderen Management-Planlet ein nicht behandelbarer Fehler auf, muss das erfolgreich ausgeführte Planlet kompensiert werden. (iv) Dazu implementiert jedes Planlet einen `bpel:compensationHandler` (CH), dessen Aufruf alle Änderungen rückgängig macht. Der Planlet-Scope inklusive dessen Handler wird bei der Plangenerierung in den dafür vorgesehenen Scope des Plans kopiert. Dadurch wird das Planlet in den Plan eingebettet.

¹In [Abbildung 5.13](#) wird die grafische Notation des *BPEL Designers* [Ecl15] verwendet.

²Die Rücksetzung der Änderungen wird durch die Invertierung des Zahnradsymbols dargestellt.

Wie in [Abschnitt 5.2.3](#) definiert bekommt jedes Planlet beim Aufruf Daten übermittelt, insbesondere eine Referenz auf das aktuelle Instanzmodell der betroffenen Anwendung sowie die statisch kompatible Zuordnung der Elemente des Topologiefragments auf die Elemente des DMMN-Modells. Dazu werden Namenskonventionen für Variablen (`bpel:variable`) definiert, die innerhalb des jeweiligen Scopes, in welchen das Planlet kopiert wird, generiert werden und von der Managementlogik des Planlets gelesen werden können. Diese Variablen werden durch generierte `bpel:assign`-Aktivitäten (nicht in [Abbildung 5.13](#) dargestellt) mit den benötigten Eingaben befüllt und diese somit auf Basis standardkonformer BPEL-Mechanismen injiziert (engl. „*content injection*“).

5.5.3 Orchestrierung heterogener Managementtechnologien durch BPEL

BPEL ermöglicht die Ausführung von Managementaktivitäten grundsätzlich durch das Konzept des Dienstaufwurfes, welches verschiedenste Interaktionsmuster ermöglicht [[BDtH05](#)]. Sollen Managementtechnologien durch ein Planlet orchestriert werden, müssen diese somit als Dienst aufrufbar sein¹. Cloud-Anbieter stellen typischerweise Managementdienste auf Basis von standardisierten Protokollen wie HTTP bzw. SOAP / HTTP zur Verfügung, wodurch deren Aufrufe in BPEL modelliert und somit direkt ausgeführt werden können² (vgl. [Abschnitt 2.3](#)). In vielen Fällen bieten Managementtechnologien diese Art der Schnittstellen jedoch nicht an, häufig aufgrund eines grundsätzlich unterschiedlichen Ausführungsparadigmas, z. B. bei deklarativen Technologien.

Um heterogene Managementtechnologien durch Planlets zu orchestrieren, können die von Wettinger et al. entwickelten Integrationsansätze auf Basis des TOSCA-Standards verwendet werden: In Wettinger et al. [[WBB+14b](#); [WBB+14a](#)] wird ein Managementbus vorgestellt, der unterschiedliche Managementtechnologien auf Basis eines Erweiterungssystems unterstützt. Der Bus ermöglicht das Aufrufen von Managementoperationen an Komponenten und Relationen einer TOSCA-Topologie, welche durch unterschiedliche Managementtechnologien implementiert sein können, und stellt dafür eine zentrale

¹Je nach Ausführungsumgebung werden unterschiedliche Protokolle und Formate unterstützt.

²Einige Workflow-Systeme, wie beispielsweise Apache ODE, benötigen hierfür spezielle Extension Activities, z. B. um REST-basierte Aufrufe auszuführen [[HFK+14](#)]

Schnittstelle, die als Dienst implementiert ist, bereit. Dieser Bus kann somit direkt von Planlets aufgerufen werden. Involvierte Instanzdaten müssen dabei vom Planlet beim Aufruf übergeben und das Resultat anschließend ins Instanzmodell übertragen werden. Zur Einbettung verschiedener Managementtechnologien in TOSCA stellen Wettinger et al. [WBL14b] einen Ansatz vor, der Metainformationen verschiedener Technologieartefakte dazu verwendet, Operationsimplementierungen für TOSCA Komponententypen (engl. „*Node Types*“) und Relationentypen (engl. „*Relationship Types*“) zu generieren. Diese können anschließend über den Managementbus aufgerufen werden.

Neben dieser zentralen Managementkomponente wurde von Wettinger et al. [WBL15a] zudem ein generischer Ansatz namens ANY2API zur Generierung von Programmierschnittstellen (engl. „*APIs*“) vorgestellt, auf Basis dessen verschiedene Managementartefakte und Technologien als aufrufbare Dienste gekapselt werden können. Der Ansatz analysiert aufzurufende Artefakte und deren Metadaten, um automatisiert eine Dienstimplementierung zu generieren, welche die gewünschten Transportprotokolle und Formate unterstützt. Dadurch können verschiedene Technologien und Artefakte über eine generierte Dienstschnittstelle indirekt durch BPEL-basierte Planlets orchestriert werden.

BPEL unterstützt neben diesen dienstbasierten Aufrufen zudem das Konzept der *Extension Activities*, welche zur Realisierung domänenspezifischer Erweiterungen eingesetzt werden können [OAS07b]. Diese Erweiterungen erlauben eine individuelle Aktivitätsimplementierung durch Programmiersprachen wie Java und ermöglichen dadurch die Integration und den Aufruf beliebiger Managementtechnologien, sofern diese durch die eingesetzte Programmiersprache umsetzbar sind. Mittels *Extension Activities* kann somit die Ausführung beliebiger Managementtechnologien direkt in BPEL modelliert werden, beispielsweise um Skripte auf virtuelle Maschinen zu kopieren und auszuführen. Ein Konzept zur direkten Integration deklarativer Managementtechnologien in imperative Workflow-Modelle wurde in Breitenbücher et al. [BBK+15a] vorgestellt und ebenfalls anhand der Sprache BPEL auf Basis von *Extension Activities* validiert. Diese Technologien stellen somit die Grundlage für die effiziente Umsetzung von Management-Planlets zur Ausführung verschiedenster Managementaufgaben.

5.6 Diskussion und geplante Forschungsarbeiten

In diesem Abschnitt werden unterschiedliche Aspekte des vorgestellten Ansatzes diskutiert sowie ein Ausblick auf zukünftige Forschungsarbeiten gegeben.

5.6.1 Kapselung externer Technologien

Das Konzept der Management-Planlets ermöglicht die nahtlose Kombination und Integration heterogener Managementtechnologien durch zwei grundlegende Ansätze: (i) Uniforme, formale Beschreibung der Managementfunktionalität und (ii) modularisierte Technologieorchestrierung. Durch DMMN kann die Funktionalität eines Planlets technologieunabhängig und formal beschrieben werden, wodurch diese sowohl funktional als auch bezüglich ihrer Vorbedingungen und Effekte sowohl miteinander verglichen als auch auf formalen Grundlagen kombiniert werden können. Dadurch können mehrere Planlets zu einem *komponierten Management-Planlet* aggregiert werden, welches eine umfangreichere Funktionalität implementiert. Daraus resultiert ein rekursives Aggregationsmodell, das ähnlich zu Workflow-Sprachen wie BPEL unterschiedliche Granularitätsstufen ermöglicht. Dieses Konzept wurde in Falkenthal et al. [FBB+14b; FBB+14a] eingesetzt, um ausführbare Workflow-Modelle als vorab implementierte Lösungen für Managementmuster zu erstellen.

Die technischen Integrationsdetails externer Technologien betreffen ausschließlich lokale Aspekte des imperativen Managementmodells eines Planlets. Die dadurch geschaffene Integrationsschicht ermöglicht somit die Entkopplung von Managementfunktionalität und technischer Integration der dazu eingesetzten Technologien, wodurch beliebige Systeme, insbesondere auch Altlasten (engl. „*legacy software*“), modularisiert in umfangreiche Managementprozesse integriert werden können. Speziell manuelle Managementaufgaben, die Menschen zu deren Ausführung erfordern, etwa zur Inbetriebnahme eines physikalischen Servers [ML08], können dadurch transparent integriert werden. Die technischen Details betreffen ausschließlich den zuständigen Planlet-Entwickler, wodurch eine Trennung der Verantwortlichkeiten ermöglicht und die Kooperation unterschiedlicher Experten unterstützt wird. Zudem ermöglicht DMMN die intuitive

Modellierung von Topologiefragmenten auf Basis bekannter Konzepte der Objektorientierung. Im Gegensatz zu anderen formalen Ansätzen erleichtert dies die Beschreibung der von Planlets realisierten Managementfunktionalität. Insbesondere haben Eilam et al. [EEKS11] festgestellt, dass die Beschreibung von Aktionen für das AI-Planning mittels PDDL Modellierer vor Herausforderungen stellt, wenn diese Objektorientierung gewohnt sind. Aufgrund der häufig in dieser Form strukturieren CMDDBs stellt dies speziell im Anwendungsmanagement eher die Regel als die Ausnahme dar (vgl. [Abschnitt 2.1.3](#)).

5.6.2 Fehlerbehandlung und Kompensation

Die Mechanismen zur Fehlerbehandlung und Kompensation ermöglichen die atomare Ausführung von Planlets, wodurch auch die generierten Pläne dieser Eigenschaft folgen. Dadurch wird sichergestellt, dass die Anwendung im Falle eines Fehlers in einen konsistenten Anwendungszustand zurückgesetzt wird und alle Effekte der Managementaufgaben bestmöglich rückgängig gemacht werden. Die Erkenntnisse von Machado et al. [MDdC+08; MdCdS+09] zur Fehlerbehandlung in Managementplänen werden durch diese Mechanismen auf die Komposition von Subprozessen übertragen (siehe [Abschnitt 2.3.7.2](#)).

Die Umkehrung der Ausführungsreihenfolge zur Kompensation bereits abgeschlossener Aktivitäten in Geschäftsprozessen, wie in der vorliegenden Arbeit auf das Anwendungsmanagement übertragen, ist jedoch generell nicht in allen Fällen optimal [GFJK03]. Wird beispielsweise die Provisionierung einer virtuellen Maschine und die Installation von Software durch einen Plan modelliert, ist die Deinstallation der Software im Rahmen einer Kompensation unnötig, da die Terminierung der virtuellen Maschine im letzten Kompensationsschritt aus Managementsicht denselben Effekt hat. Diesbezügliche Optimierungsmöglichkeiten werden in der vorliegenden Arbeit nicht untersucht, können jedoch in Schritt 5 der PALMA-Methode manuell durch Adaption des generierten Managementplans umgesetzt werden. Ansätze zur Steigerung der Kompensationseffizienz, wie beispielsweise von Weber et al. [WWF+12; WWF+13] vorgestellt (siehe [Abschnitt 2.3.7.2](#)), sind Teil zukünftiger Forschungsarbeiten.

5.6.3 Alterung generierter Managementpläne

Managementpläne unterliegen, wie in [Abschnitt 2.3.5.3](#) diskutiert, Alterungseffekten. Da diese durch die PALMA-Methode für das aktuelle Instanzmodell der betroffenen Anwendung generiert werden, reduzieren sich mögliche externe Einflüsse. Zudem können Planlets vor Durchführung der Managementaufgaben den aktuellen Zustand der betroffenen Komponenten bzw. Relationen mit dem des DMMN-Modells abgleichen¹, um im Falle einer Diskrepanz die Ausführung abubrechen und die Ausführung gemäß [Abschnitt 5.2.4](#) rückgängig zu machen.

Zur Erneuerung gealterter Prozessmodelle stellen Hagen und Kemper [[HK10b](#)] drei Konzepte vor: (i) Erneute Planung (engl. „*replanning*“), (ii) Anpassung des Plans (engl. „*plan adaptation*“) und (iii) Anpassung der Planparameter (engl. „*parameter adaptation*“). Die erneute Planung wird durch den Ansatz dahingehend unterstützt, dass DMMN-Modelle automatisiert durch Anwendung automatisierter Managementmuster generiert werden können. Die PALMA-Methode unterstützt zudem die manuelle Anpassung des generierten Plans. Eine Anpassung der Planparameter ist für den vorgestellten Ansatz nicht möglich, da die generierten Pläne Informationen ausschließlich über das Instanzmodell der Anwendung austauschen (siehe [Abschnitt 5.2.3](#)). Die Automatisierung der Neuplanung im Falle eingetretener Alterungseffekte ist aufgrund der durch Management-Planlets geschaffenen Modularität von Managementworkflows ein aussichtsreicher Ansatzpunkt zukünftiger Forschungsarbeiten.

5.6.4 Berechnungskomplexität der Plangenerierung

Der vorgestellte Algorithmus zur Generierung von Managementplänen basiert auf Partial Order Planning [[Wel94](#)] und trifft nicht-deterministische Entscheidungen, um Planlets zur Ausführung von Annotationen auszuwählen sowie um deren Ausführungsreihenfolge zu bestimmen. Diese nicht-deterministischen Entscheidungen werden in der Implementierung mittels Tiefensuche und Back-

¹Einige Systeme zur Erfassung aktueller Anwendungszustände, wie beispielsweise das ETG-Framework und typischerweise CMDBs (siehe [Abschnitt 2.1.3](#)), unterstützen das Aktualisieren von Instanzmodellen hinsichtlich einzelner Komponenten. Aktuelle Informationen können somit gezielt von Planlets zum Ausführungszeitpunkt abgerufen werden.

tracking in deterministische Anweisungen übersetzt, wodurch der Algorithmus unter einer exponentiellen Berechnungskomplexität leidet (siehe [Algorithmus 5.1](#)). Die Optimierung dieser Komplexität ist jedoch nicht Gegenstand der vorliegenden Arbeit, da diese aufgrund der Charakteristika der Managementdomäne keine praktische Relevanz erfährt und trotz der theoretischen Komplexität eine effiziente und schnelle Berechnung der Pläne ermöglicht:

Die Ausführung einer Managementannotation wird typischerweise stark von der jeweiligen betroffenen Komponente bzw. Relation beeinflusst, wodurch für verschiedene Topologiekonstellationen unterschiedliche Planlets erforderlich sind. Wie in [Abschnitt 5.4.5.3](#) diskutiert unterscheidet sich z. B. die Installation einer bestimmten Software auf einem Linux-basierten System fundamental von deren Installation auf einem Windows-System, wodurch die Kombination aus Komponenten, Relationen und Managementannotationen die Menge möglicher Planlets signifikant einschränkt. Im Regelfall kann eine bestimmte Managementannotation nur durch ein einziges Management-Planlet ausgeführt werden, wodurch sich dessen Selektion effizient deterministisch berechnen lässt und sich der Berechnungsaufwand in der Praxis in Grenzen hält. Speziell bei der Bereitstellung von Komponenten bzw. der Installation von Software ist dies der generelle Fall [[EME+06](#)]. Die Entwicklung von Management-Planlets sollte daher diese Eigenschaft berücksichtigen und semantisch identische Funktionalität bezüglich einer bestimmten Topologiestruktur nicht mehrfach durch verschiedene Planlets implementieren, sondern diese im Sinne der Modularisierung in einem Planlet zusammenfassen. In ähnlicher Weise lässt sich die Ausführungsreihenfolge von Managementaufgaben meistens eindeutig bestimmen, sodass nur in Sonderfällen ein Backtracking erforderlich ist, d. h. wenn keine direkten Abhängigkeiten zwischen zwei Managementaufgaben existieren.

5.6.5 Abgrenzung zu verwandten Arbeiten

Da Silva et al. [[dSdL11](#)] stellen eine generische, AI-Planning-basierte Methode und Systemarchitektur zur automatisierten Generierung von Prozessen auf Basis von PDDL vor und wenden diese exemplarisch auf das Anwendungs-

management an. Der generelle Ansatz, die Plangenerierung in die Phasen „Planung“ und „Transformation in Workflow“ zu unterteilen, wird in der vorliegenden Arbeit aufgegriffen und ebenso in der Architektur (siehe [Abschnitt 8.1](#)) reflektiert. Die Umsetzung der Phasen unterscheidet sich jedoch grundlegend.

Eilam et al. [[EEKS11](#)] setzen zur Generierung von Bereitstellungs-Workflows aus Zielzustandsmodellen ebenfalls Subgraphisomorphismen ein, um die Anwendbarkeit der Aktionen zu evaluieren. Das DMMN-Metamodell und damit die Topologiefragmente von Planlets ermöglichen jedoch weitaus spezifischere Vorbedingungen zu spezifizieren, beispielsweise die Verfügbarkeit bestimmter Operationen oder Artefakte. Des Weiteren können Anforderungen und Effekte bezüglich Relationen in DMMN modelliert werden, was alle in [Abschnitt 2.3.3](#) vorgestellten existierende Ansätze nicht unterstützen. Der grundlegende Unterschied liegt jedoch in der Art Managementziele zu spezifizieren, die in DMMN explizit durch Managementannotationen modelliert werden können. Dadurch ermöglicht das Konzept der vorliegenden Arbeit neben zustandsmodifizierenden auch zustandserhaltende Managementaufgaben zu modellieren und auszuführen. Durch Kombination dieser Technik mit AI-Planning können so, im Gegensatz zum Ansatz von Eilam et al. [[EEKS11](#)], umfangreiche Managementaufgaben modelliert und ausgeführt werden, die unter Umständen mehrfache Zustandsänderungen einer Komponente oder Relation verursachen.

Im Gegensatz zum Ansatz von Levanti und Ranganathan [[LR09](#)], in welchem Konfigurationsziele durch eine flache Liste an Tags abstrakt modellierbar sind, ermöglicht DMMN die Deklaration der abstrakten Managementziele direkt an den betroffenen Komponenten und Relationen. Während beide Ansätze die Abstraktion feingranularer Konfigurationen ermöglichen, ist ein grundlegender Unterschied, dass auch zustandserhaltende Managementaufgaben in DMMN modelliert und durch Management-Planlets ausgeführt werden können.

Der Einsatz von Subprozessen zur Realisierung unterschiedlicher Prozessgranularität wurde von Mietzner und Leymann [[ML08](#)] in ähnlicher Form für die Provisionierung von Anwendungen verwendet, jedoch ohne formale deklarative Beschreibung der implementierten Managementfunktionalität.

5.6.6 Zukünftige Forschungsarbeiten

Planlets werden in dieser Arbeit als Inline Sub-process in den generierten Plan eingebettet (siehe [Abschnitt 5.5.1](#)), weshalb Plan und Planlets in derselben Sprache modelliert sein müssen bzw. die Sprache des Plans die Sprache des Planlets unterstützen muss. Um diese Abhängigkeit aufzuheben, können Planlets als entfernte Subprozesse realisiert werden (engl. „*Remote Sub-process*“ [[LR00](#)]). Dies erfordert speziell bei der Umsetzung mit BPEL besondere Anstrengungen, da BPEL nicht den direkten Aufruf entfernter Subprozesse unter Einhaltung der geforderten Lebenszyklusabhängigkeiten unterstützt. Die BPEL-Erweiterung *WS-BPEL Extension for Sub-processes (BPEL-SPE)* [[KKL+05](#)] löst dieses Problem zwar durch Einführung von Konstrukten zum expliziten Aufruf entfernter Subprozesse, wird jedoch nicht von allen BPEL-Ausführungsumgebungen unterstützt, wodurch die angestrebte Portabilität nicht gegeben ist. Die fehlende Unterstützung verteilter traditioneller Subprozesse in BPEL kann jedoch durch Kombination standardkonformer BPEL-Konstrukte approximiert werden.

Vanhatalo et al. [[VVL07](#); [VVK08](#)] stellen ein Verfahren vor, um SESE-Fragmente in Workflow-Modellen zu erkennen. Die Analyse existierender Management-Workflows und die Extraktion identisch auftretender Subprozesse legt die Grundlage zur semi-automatisierten Generierung neuer Planlets, die bewährte Managementlogik wiederverwendbar und automatisiert ausführbar erfassen. Des Weiteren wurden Planlets in der vorliegenden Arbeit als SESE-Prozesse modelliert, was die Komposition von Planlets zu Managementplänen vereinfacht, jedoch die Extraktion möglicher wiederverwendbarer Prozessfragmente aus existierenden Modellen einschränkt. Die Arbeiten von Eberle et al. [[EUL09](#)] und Schumm et al. [[SKK+11](#)] untersuchen die Wiederverwendung und Komposition von Prozessfragmenten, die nicht notwendigerweise korrekt und vollständig modellierte Prozessteile darstellen, d. h. beispielsweise Kontrollflusskanten ohne Zielaktivität beinhalten (engl. „*dangling references*“). Die Lockerung der Planlet-Definition hinsichtlich dieser Art Fragmente würde die Anzahl wiederverwendbarer Prozessteile deutlich erhöhen, da das einschränkende SESE-Kriterium entfällt. Zur Verwaltung und Versionierung von Planlets kann zudem beispielsweise das Repository *Fragmento* [[SKLS10](#)] integriert werden.

KAPITEL 6

AUTOMATISIERTE ANWENDUNG VON MANAGEMENTMUSTERN

Nachdem in [Kapitel 5](#) mit dem Konzept der Management-Planlets ein generischer Beitrag zur Automatisierung technischer Managementlösungen vorgestellt wurde, betrachtet dieses Kapitel die Automatisierung konzeptionellen, abstrakten Managementwissens auf Basis des Musteransatzes. Dazu wird das Konzept der *automatisierten Managementmuster* vorgestellt, welche Musterlösungen automatisiert auf Instanzmodelle von Anwendungen anwenden und diese in deklarative DMMN-Modelle transformieren. Managementmuster werden dabei auf verschiedenen Abstraktionshöhen diskutiert und als semi- bzw. vollautomatisierte Transformationen realisiert. Der vorgestellte Automatisierungsansatz trägt somit zur Realisierung des zweiten Schritts der PALMA-Methode ([Kapitel 3](#)) bei und stellt [Forschungsbeitrag 4](#) der vorliegenden Dissertation dar.

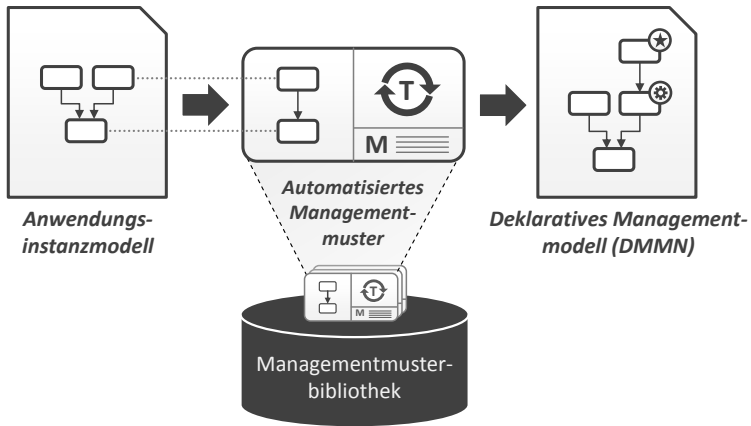


Abbildung 6.1: Grundkonzept der Automatisierung von Managementmustern

6.1 Idee und Grundkonzept

Die zentrale Idee dieses Forschungsbeitrags ist die generische Automatisierung bewährten Managementwissens auf unterschiedlichen Abstraktionsebenen, um Managementprobleme, wie beispielsweise die Migration einer Anwendung ohne Ausfallzeit in eine Cloud-Umgebung, unabhängig von individuellen Anwendungen zu lösen und automatisiert im Rahmen der PALMA-Methode auszuführen. In diesem Kapitel wird dazu das Konzept der automatisierten Managementmuster eingeführt, welches (i) Kontext und Problem des Managementmusters in Form eines DMMN-basierten Topologiefragments formalisiert und (ii) dessen Lösung mittels einer ausführbaren Transformation automatisiert.

Abbildung 6.1 zeigt eine Darstellung dieser Idee. Automatisierte Managementmuster überführen Anwendungsinstanzmodelle, deren Struktur der formalen Beschreibung von Kontext und Problem des Musters entsprechen, automatisiert in deklarative DMMN-Modelle. Die resultierenden Modelle spezifizieren deklarativ alle für die jeweilige Anwendung auszuführenden Managementaufgaben der Musterlösung in Form abstrakter Managementannotationen.

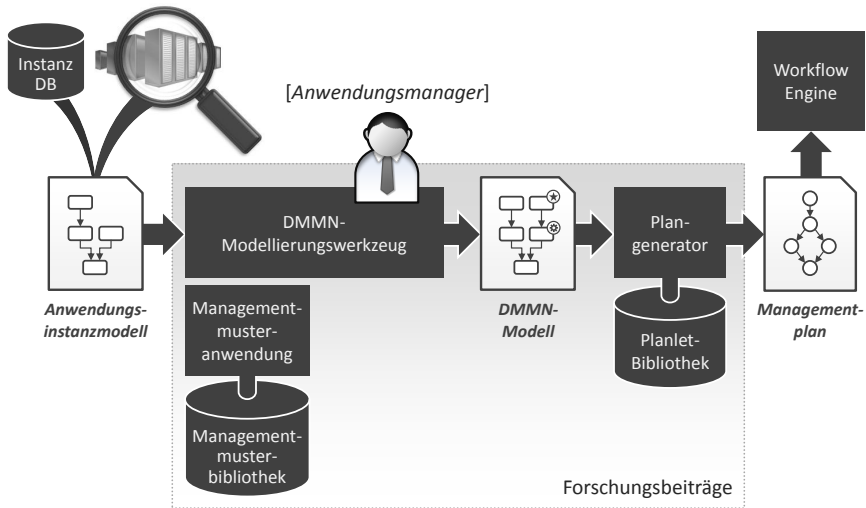


Abbildung 6.2: Erweiterung des PALMA-Frameworks um Musterunterstützung

In [Abbildung 6.2](#) ist dieser Ansatz zur automatisierten Anwendung von Managementmustern im Rahmen des in der vorliegenden Arbeit entwickelten PALMA-Frameworks dargestellt. Während die vorhergehenden Forschungsbeiträge die Modellierung von DMMN-Modellen ([Kapitel 4](#)) sowie deren vollautomatische Transformation in ausführbare Managementpläne ermöglichen ([Kapitel 5](#)), erweitert dieser Forschungsbeitrag das Modellierungswerkzeug um die Musterunterstützung. Dadurch können Anwendungsmanager automatisierte Managementmuster für Instanzmodelle betroffener Anwendungen ausführen, um DMMN-Modelle zur Lösung eines bestimmten Managementproblems zu generieren, welche anschließend durch den zweiten Forschungsbeitrag automatisiert in ausführbare Managementpläne transformiert und ausgeführt werden.

Die Erweiterung des PALMA-Frameworks um die Musteranwendung ermöglicht die Automatisierung des zweiten Schritts der PALMA-Methode sowie die manuelle bzw. automatisierte Verfeinerung angewendeter Muster in Schritt 3 durch das DMMN-Modellierungswerkzeug. Die Architektur und prototypische Realisierung des Forschungsbeitrags wird detailliert in [Kapitel 8](#) vorgestellt.

6.2 Automatisierte Managementmuster

In diesem Abschnitt wird das Konzept der automatisierten Managementmuster eingeführt, welches die (i) generische Erfassung konzeptioneller Managementexpertise sowie (ii) deren automatisierte Ausführung für individuelle Anwendungen ermöglicht, um wiederkehrende Managementprobleme zu lösen.

Automatisierte Managementmuster lösen bekannte Managementprobleme auf unterschiedlichen Abstraktionsebenen, beispielsweise um Dienstmigrationsmuster [FLR+13], Muster zum Management von Cloud-Anwendungen [FLRS12; FLR+14] oder Muster zur Optimierung des ökologischen Einflusses von Anwendungen [NLS+11] zu automatisieren. Ein automatisiertes Managementmuster besteht aus einer Transformation, die auf einer formal definierten Anwendungsstruktur angewendet werden kann, um ein Anwendungsinstanzmodell, auf das diese Struktur abgebildet werden kann, in ein deklaratives Managementmodell zu überführen. Das resultierende Modell beschreibt alle Managementaufgaben, die zur Umsetzung der Musterlösung auf der definierten Anwendungsstruktur auszuführen sind. Automatisierte Managementmuster modellieren Lösungen somit rein deklarativ, d. h. ohne technische Details zu spezifizieren, wie die Managementziele zu erreichen sind. Dadurch erfasst die Transformation die Musterlösung ausschließlich auf konzeptioneller Abstraktionsebene und somit unabhängig von konkreten Managementtechnologien und technischer Ausführungslogik. Durch die formale Definition der Anwendungsstrukturen, auf die ein Managementmuster angewendet werden kann, bleibt dessen Transformation unabhängig von individuellen Anwendungen und kann generisch auf alle Anwendungen angewendet werden, welche dieser Struktur entsprechen.

Definition 71 (Automatisiertes Managementmuster - informell). Ein *automatisiertes Managementmuster* ist eine generische, automatisiert ausführbare Transformation, die ein bestimmtes Managementproblem bezüglich einer laufenden Anwendung durch Deklaration auszuführender Managementaufgaben löst. Vorbedingungen zur Ausführung dieser Transformation werden formal in Form der Anwendungsstruktur definiert, auf welche das automatisierte Managementmuster angewendet werden kann. ■

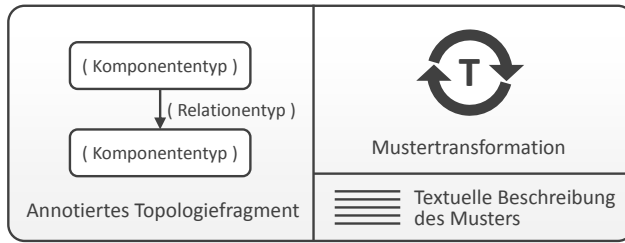


Abbildung 6.3: Aufbau eines automatisierten Managementmusters

6.2.1 Konzeptioneller Aufbau eines automatisierten Managementmusters

Ein automatisiertes Managementmuster besteht konzeptionell aus drei Bestandteilen: (i) Spezifikation der Vorbedingungen zur Anwendung des Musters, (ii) Transformation, welche die Musterlösung automatisiert anwendet, sowie (iii) einer textuellen Beschreibung des Managementmusters.

Die Spezifikation der Vorbedingungen wird, ähnlich wie bei Management-Planlets, durch ein *Topologiefragment* realisiert, welches die Topologiestruktur, auf die das Muster angewendet werden kann, formal in Form eines DMMN-Modells beschreibt. Durch das Topologiefragment ist ein automatisiertes Muster ausschließlich an die definierte Teilstruktur gekoppelt und kann somit auf sämtlichen Anwendungen ausgeführt werden, deren Instanzmodelle dieses Topologiefragment als kompatiblen Teilgraph gemäß der Kompatibilitätsdefinition $\approx_M$, welche in [Abschnitt 6.3.1](#) vorgestellt wird, enthalten. Automatisierte Managementmuster dürfen ebenfalls Managementannotationen als Vorbedingungen spezifizieren, beispielsweise um deren Ausführung zu optimieren.

Der zweite Bestandteil stellt die Transformation dar, die das Instanzmodell der Anwendung konsumiert und dieses gemäß der Musterlösung transformiert, z. B. Komponenten und Relationen hinzufügt und Managementannotationen deklariert. Generelle Regeln und Einschränkungen, welche bei der Transformation Beachtung finden müssen, werden in [Abschnitt 6.3.2](#) vorgestellt.

Der dritte Bestandteil stellt die textuelle Beschreibung des Musters dar, die im typischen Musterformat dokumentiert ist (vgl. [Abschnitt 2.4](#)).

6.2.2 Formales Metamodell automatisierter Managementmuster

In diesem Abschnitt wird das formale Metamodell für automatisierte Managementmuster vorgestellt. Um die Variante der Methode zur Kombination mehrerer Muster zu unterstützen (siehe [Abschnitt 3.4](#)), müssen automatisierte Managementmuster neben (i) Instanzmodellen auch auf (ii) deklarative Managementmodelle angewendet werden können, die von anderen Mustern generiert wurden. Um formal einen einheitlichen Ansatz zu gewährleisten, werden automatisierte Managementmuster auf Basis von DMMN definiert, da Instanzmodelle per [Definition 11](#) in deklarativen Managementmodellen enthalten sind und DMMN somit auch zur Beschreibung reiner Anwendungsinstanzen eingesetzt werden kann. Im Rahmen des PALMA-Frameworks werden andere Formate von Instanzmodellen daher explizit in DMMN transformiert ([Abschnitt 8.1.1](#)).

Definition 72 (Automatisiertes Managementmuster). Die Menge aller automatisierten Managementmuster ist definiert als AMM . Sei $m \in AMM$ ein automatisiertes Managementmuster, dann ist m definiert als Tripel:

$$AMM \subseteq D \times T \times \Sigma^+$$

$$m = (tf_m, t_m, b_m)$$

Dabei ist

- $tf_m \in D$ das DMMN-Topologiefragment, das die Anwendungsstruktur spezifiziert, auf die das Muster angewendet werden kann (siehe [Abschnitt 6.3.1](#)),
- $T := \{t \mid t: D \times (\mathfrak{M}^s \times \mathfrak{M}^{ab}) \rightarrow D \cup \{\perp\}\}$ die Menge aller Transformationen aller automatisierten Muster,
- $t_m \in T$ eine Abbildung, welche die Lösung des Managementmusters als automatisierte Transformation beschreibt. Die Abbildung transformiert DMMN-Modelle $\{d_1, \dots, d_n\} \in D$, auf die tf_m durch eine statisch kompatible Zuordnung $(m^s, m^{ab}) \in \mathfrak{M}^s \times \mathfrak{M}^{ab}$ abgebildet werden kann, in DMMN-Modelle $\{d_1^m, \dots, d_n^m\} \in D$, welche die Musterlösung für die jeweiligen Anwendungen in Form von Annotationen deklarieren (siehe [Abschnitt 6.3.2](#)).
- b_m die nicht-leere textuelle Beschreibung des Musters. ■

6.3 Anwendung automatisierter Managementmuster

Die Ausgangslage zur Anwendung eines automatisierten Managementmusters ist durch die Existenz des Instanzmodells der Anwendung gegeben, für die ein Managementproblem gelöst werden soll. Die PALMA-Methode definiert dazu vor der Anwendung von Mustern die Beschaffung des aktuellen Anwendungsinstanzmodells (siehe [Abschnitt 3.3.2](#)). In diesem Abschnitt wird der darauffolgende zweite Schritt der PALMA-Methode beschrieben, in dem ein Managementmuster selektiert und angewendet wird. Die Anwendung eines automatisierten Managementmusters umfasst daher zwei Schritte: (i) Überprüfung der Anwendbarkeit eines automatisierten Managementmusters und (ii) Ausführung dessen Mustertransformation. Dazu wird in [Abschnitt 6.3.1](#) eine formale Kompatibilitätsrelation definiert und ein Algorithmus vorgestellt, mithilfe derer anwendbare Managementmuster identifiziert werden können. Anschließend werden in [Abschnitt 6.3.2](#) Mustertransformationen, deren Ein- und Ausgaben sowie die Regeln, denen sie unterliegen, im Detail beschrieben.

6.3.1 Berechnung anwendbarer Managementmuster

In diesem Abschnitt wird ein Algorithmus vorgestellt, mithilfe dessen alle automatisierten Managementmuster gefunden werden können, die zum Management einer bestimmten Anwendung ausgeführt werden können. Automatisierte Managementmuster spezifizieren ihre Anwendbarkeit gleichermaßen wie Management-Planlets durch ein annotiertes Topologiefragment. Daher gelten die in [Abschnitt 5.4.2](#) definierten Kompatibilitätsrelationen als Ausgangslage zur Evaluation, ob das Topologiefragment eines Musters zu einem bestimmten DMMN-Modell kompatibel ist, d. h. ob das Muster auf die repräsentierte Anwendung anwendbar ist. Während die dynamische Kompatibilität ([Abschnitt 5.4.2](#)) zwischen Topologiefragment und DMMN-Modell für Planlets erst während der Plangenerierung evaluiert wird (vgl. [Abschnitt 5.4.7.1](#)), werden zustandsbezogene Vorbedingungen automatisierter Managementmuster gleichzeitig zu deren statischen Vorbedingungen überprüft. Dies resultiert aus dem Vorgehensmodell der PALMA-Methode: Automatisierte Managementmuster werden manuell

vom Anwendungsmanager für eine bestimmte Anwendungsinstanz ausgewählt, wozu deren dynamische Kompatibilität mit dem zugehörigen Instanzmodell direkt bei der Selektion überprüft werden muss (siehe [Abschnitt 3.3.3](#)). Sollen mehrere Muster iterativ angewendet werden, werden diese ebenfalls manuell in der Variante der Methode selektiert (vgl. [Abschnitt 3.4](#)). Sie werden somit in der vorliegenden Arbeit nicht automatisiert miteinander kombiniert, was jedoch Gegenstand zukünftiger Forschungsarbeiten darstellt. Zur Evaluation der dynamischen Kompatibilität muss daher zunächst die Kompatibilität zwischen Eigenschaftenelementen zweier DMMN-Modelle definiert werden.

6.3.1.1 Kompatibilität von Eigenschaften

Ein Eigenschaftenelement in einem Topologiefragment eines automatisierten Managementmusters ist kompatibel zu einem Eigenschaftenelement eines DMMN-Modells, wenn beide Elemente bezüglich der Attribute „Name“ und „Datentyp“ übereinstimmen. Falls die geforderte Eigenschaft auf Musterseite einen konkreten „Wert“ spezifiziert, muss dieser zum Wert der zu vergleichenden Eigenschaft kompatibel sein. Definiert die Mustereigenschaft einen beliebigen Wert (ausgedrückt durch „*“), so muss die gleichnamige Eigenschaft im DMMN-Modells einen beliebigen Wert spezifizieren und darf nicht undefiniert (ausgedrückt durch „ $\perp$ “) sein. Ist der Wert der auf Musterseite geforderten Eigenschaft undefiniert, muss das DMMN-Modell keinen Wert für die entsprechende Eigenschaft spezifizieren. Zur klaren Unterscheidung stellt d_1 in diesem Kapitel durchgehend das Topologiefragment des Musters dar, d_2 das DMMN-Modell, für welches die Anwendbarkeit des Musters evaluiert werden soll.

Definition 73 (Kompatibilität von Eigenschaftenelementen). Seien $e_1 \in E_{d_1}$ und $e_2 \in E_{d_2}$ Eigenschaftenelemente zweier DMMN-Modelle $d_1, d_2 \in D$, dann ist die Kompatibilitätsrelation $\approx_e \subseteq E \times E$ definiert als:

$$e_1 \approx_e e_2 :\Leftrightarrow \pi_2(e_1) = \pi_2(e_2) \wedge \pi_4(e_1) = \pi_4(e_2) \wedge (\pi_3(e_1) = \pi_3(e_2) \vee (\pi_3(e_1) = * \wedge \pi_3(e_2) \neq \perp) \vee \pi_3(e_1) = \perp)$$

■

6.3.1.2 Dynamische Kompatibilität zwischen DMMN-Modellen

Auf Basis der im vorangegangenen Abschnitt definierten Kompatibilitätsdefinition bezüglich Eigenschaften werden dynamisch kompatible Zuordnungen zwischen DMMN-Modellen wie folgt definiert:

Definition 74 (Dynamisch kompatible Zuordnung). Seien $d_1, d_2 \in D$ zwei DMMN-Modelle, dann ist d_1 bezüglich einer statisch kompatiblen Zuordnung $(m^s, m^{ab}) \in \mathfrak{M}_{d_1, d_2}^{\approx_s}$ genau dann zustandsbezogen zu d_2 kompatibel, wenn die beiden Modelle sowie deren Zuordnung bezüglich der vierstelligen Relation $kompatibel^\bullet \subseteq D \times D \times \mathfrak{M}^s \times \mathfrak{M}^{ab}$ wie folgt in Beziehung zueinander stehen:

$$(d_1, d_2, m^s, m^{ab}) \in kompatibel^\bullet : \Leftrightarrow (m^s, m^{ab}) \in \mathfrak{M}_{d_1, d_2}^{\approx_s} \wedge \\ (\forall (te_{d_1}, te_{d_2}) \in m^s \forall e_{d_1} \in eigeneEigenschaften_{d_1}(te_{d_1}) \\ \exists e_{d_2} \in eigenschaften(te_{d_2}): e_{d_1} \approx_e e_{d_2})$$

Sei das Quadrupel $(d_1, d_2, m^s, m^{ab}) \in kompatibel^\bullet$, dann wird das Paar (m^s, m^{ab}) als *dynamisch kompatible Zuordnung* bezeichnet. ■

Dynamisch kompatible Zuordnungen berücksichtigen dabei nur die eigenen Eigenschaften der Topologieelemente des DMMN-Modells d_1 , welches das Topologiefragment des Musters darstellt, um dessen Vorbedingungen präzise definieren zu können. Geerbte Eigenschaften, die nicht als Vorbedingungen berücksichtigt werden sollen, werden durch dieses Vorgehen ignoriert.

Die dynamische Kompatibilität zweier DMMN-Modelle wird daher mit der Existenz mindestens einer dynamisch kompatiblen Zuordnung zwischen den Topologie- und Annotationenelementen beider DMMN-Modelle gleichgesetzt:

Definition 75 (Dynamische Kompatibilität von DMMN-Modellen). Seien $d_1 \in D$ und $d_2 \in D$ DMMN-Modelle, dann ist d_1 *dynamisch* zu d_2 kompatibel genau dann wenn diese bezüglich der Kompatibilitätsrelation $\tilde{\approx}_\bullet \subseteq D \times D$ wie folgendermaßen definiert in Beziehung zueinander stehen:

$$d_1 \tilde{\approx}_\bullet d_2 : \Leftrightarrow \exists m^s \in \mathfrak{M}^s \exists m^{ab} \in \mathfrak{M}^{ab} : kompatibel^\bullet(d_1, d_2, m^s, m^{ab}) \quad \blacksquare$$

Die Menge aller dynamisch kompatiblen Zuordnungen zwischen zwei DMMN-Modellen wird wie folgt definiert:

Definition 76 (Menge aller dynamisch kompatiblen Zuordnungen). Sei $d_1 \in D$ und $d_2 \in D$ jeweils ein DMMN-Modell, dann enthält die Menge $\mathfrak{M}_{d_1, d_2}^{\approx \bullet}$ alle dynamisch kompatiblen Zuordnungen zwischen d_1 und d_2 :

$$\mathfrak{M}_{d_1, d_2}^{\approx \bullet} := \{(m^s, m^{ab}) \in \mathfrak{M}^s \times \mathfrak{M}^{ab} \mid (d_1, d_2, m^s, m^{ab}) \in \text{kompatibel}^\bullet\}$$

■

6.3.1.3 Kompatibilität von Managementmustern und DMMN-Modellen

Auf Basis der formal definierten dynamischen Kompatibilität zwischen DMMN-Modellen wird die Anwendbarkeit eines automatisierten Managementmusters auf ein bestimmtes DMMN-Modell $d \in D$ wie folgt formal definiert:

Definition 77 (Kompatibilität von Managementmustern und DMMN-Modellen). Sei $m \in AMM$ ein automatisiertes Managementmuster, $tf_m := \pi_1(m)$ dessen Topologiefragment und $d \in D$ ein DMMN-Modell, dann wird die Kompatibilitätsrelation $\tilde{\approx}_M$ zur Bestimmung der Anwendbarkeit des Musters m auf die von d repräsentierte Anwendung wie folgt formal definiert:

$$\begin{aligned} \tilde{\approx}_M &\subseteq AMM \times D \\ m \tilde{\approx}_M d &:\Leftrightarrow tf_m \tilde{\approx}_\bullet d \end{aligned}$$

■

Dementsprechend ist m auf d anwendbar, wenn mindestens eine dynamisch kompatible Zuordnung der Elemente des annotierten Topologiefragments tf_m des Musters m auf die Elemente des DMMN-Modells d existiert.

6.3.1.4 Algorithmus zur Überprüfung der Anwendbarkeit eines Musters

Gemäß Definition 77 ist ein Muster $m \in AMM$ genau dann auf ein DMMN-Modell $d \in D$ anwendbar, wenn mindestens eine dynamisch kompatible Zuordnung zwischen dem Topologiefragment tf_m und dem DMMN-Modell existiert. Daher wird zunächst die Berechnung dynamisch kompatibler Zuordnungen definiert: Algorithmus 6.1 überprüft für alle statisch kompatiblen Zuordnungen zwischen dem Topologiefragment und dem DMMN-Modell (Zeile 4), ob diese auch dynamisch kompatibel (Zeilen 7–15) sind und liefert sie im Erfolgsfall zurück.

Algorithmus 6.1 BerechneKompatibleZuordnungen $\tilde{\cdot}$ (tf, d)

```

1:  $\mathcal{M}_{tf,d}^{\tilde{\cdot}} := \emptyset$ 
2:  $\mathcal{M}_{tf,d}^{\tilde{\cdot}} := \text{BerechneKompatibleZuordnungen}^{\tilde{\cdot}}(tf, d)$ 
3: // Überprüfe jede statisch kompatible Zuordnung auf deren dynamische Kompatibilität
4: for all  $((m^s, m^{ab}) \in \mathcal{M}_{tf,d}^{\tilde{\cdot}})$  do
5:    $c := \text{true}$ 
6:   // Überprüfe die zustandsbezogenen Vorbedingungen jedes Topologieelements
7:   for all  $((te_{tf}, te_d) \in m^s)$  do
8:     // Für jede eigene Eigenschaft des Topologieelements des Fragments muss es
9:     // eine kompatible Eigenschaft des zugeordneten Topologieelements geben
10:     $E_{te_{tf}} := \text{eigeneEigenschaften}_{tf}(te_{tf})$ 
11:    if  $(\neg(\forall e_i \in E_{te_{tf}} \exists e_j \in \text{eigenschaften}(te_d): e_i \tilde{\approx}_e e_j))$  then
12:      // Falls nicht, ist die dynamische Kompatibilität nicht gegeben
13:       $c := \text{false}$ 
14:    end if
15:  end for
16:  // Falls es für jede Eigenschaft jedes Topologieelements einen kompatiblen Gegenpart
17:  // gibt, ist die statisch kompatible Zuordnung auch dynamisch kompatibel
18:  if  $(c = \text{true})$  then
19:     $\mathcal{M}_{tf,d}^{\tilde{\cdot}} := \mathcal{M}_{tf,d}^{\tilde{\cdot}} \cup \{(m^s, m^{ab})\}$ 
20:  end if
21: end for
22: return  $\mathcal{M}_{tf,d}^{\tilde{\cdot}}$ 

```

Basierend auf diesem Algorithmus berechnet [Algorithmus 6.2](#) die Anwendbarkeit eines automatisierten Managementmusters $m \in AMM$ bezüglich eines DMMN-Modells $d \in D$: Das Muster ist genau dann anwendbar, wenn es mindestens eine dynamisch kompatible Zuordnung der Elemente des annotierten Topologiefragments des Musters auf die Elemente des DMMN-Modells gibt.

Algorithmus 6.2 BerechneMusteranwendbarkeit(m, d)

- 1: // Berechne alle dynamisch kompatiblen Zuordnungen zwischen tf_m und d
 - 2: $\mathfrak{M}_{tf_m, d}^{\approx \bullet} := \text{BerechneKompatibleZuordnungen}^{\approx \bullet}(tf_m, d)$
 - 3: // Das Muster m ist auf d anwendbar, wenn es mindestens eine dynamisch kompatible
 - 4: // Zuordnung zwischen dem Topologiefragment und dem DMMN-Modell gibt
 - 5: **return** ($\mathfrak{M}_{tf_m, d}^{\approx \bullet} \neq \emptyset$)
-

[Algorithmus 6.3](#) berechnet alle auf ein DMMN-Modell $d \in D$ anwendbaren automatisierten Managementmuster. Der Algorithmus iteriert über alle verfügbaren Managementmuster $m \in AMM$ ([Zeile 2](#)) und überprüft für jedes, ob es auf das eingegebene DMMN-Modell $d \in D$ anwendbar ist ([Zeile 3](#)). Im positiven Fall wird das Managementmuster in die Menge der anwendbaren Muster aufgenommen ([Zeile 4](#)). Jedes Muster in der zurückgegebenen Menge kann dabei auf mehrere Subgraphen des DMMN-Modells anwendbar sein. Die möglichen Zuordnungen der Elemente des Muster-Topologiefragments zu Topologieelementen des DMMN-Modells können durch [Algorithmus 6.1](#) bestimmt werden.

Algorithmus 6.3 BerechneAnwendbareMuster(d)

- 1: muster := $\emptyset$
 - 2: **for all** ($m \in AMM$) **do**
 - 3: **if** ($\text{BerechneMusteranwendbarkeit}(m, d)$) **then**
 - 4: $muster := muster \cup \{m\}$
 - 5: **end if**
 - 6: **end for**
 - 7: **return** muster
-

Die in diesem Kapitel vorgestellten Algorithmen werden im Rahmen des PALMA-Frameworks prototypisch implementiert, siehe [Kapitel 8](#).

6.3.2 Automatisierte Transformation zur Anwendung der Musterlösung

Ist ein Managementmuster $m \in AMM$ auf ein DMMN Modell $d \in D$ anwendbar, d. h. $m \tilde{\approx}_M d$, dann existiert eine nicht-leere Menge $\mathfrak{M}_{tf_m, d}^{\tilde{\approx} \bullet}$ dynamisch kompatibler Zuordnungen der Elemente des Topologiefragments auf Elemente in d , welche durch *BerechneKompatibleZuordnungen* $^{\tilde{\approx} \bullet}(tf_m, d)$ bestimmt wird (siehe [Algorithmus 6.1](#)). Jede Zuordnung $(m^s, m^{ab}) \in \mathfrak{M}_{tf_m, d}^{\tilde{\approx} \bullet}$ repräsentiert somit eine mögliche Musteranwendung auf die zugeordneten Elemente des DMMN-Modells d . Um die Transformation diesbezüglich zu konfigurieren und die betroffenen Topologieelemente des DMMN-Modells, auf welche das Muster angewendet werden soll, zu spezifizieren, wählt der Anwendungsmanager genau eine dieser Zuordnungen als Eingabe für die Mustertransformation aus.

Definition 78 (Transformation zur Automatisierung der Musterlösung). Sei $m \in AMM$ ein automatisiertes Managementmuster, dann wird dessen automatisierte Transformation zur Deklaration der Musterlösung als Abbildung realisiert, welche für einen eingeschränkten Definitionsbereich DMMN-Modelle generiert. Dieser ergibt sich aus der Menge $D_{tf_m}^{\tilde{\approx} \bullet}$ und der Menge $\mathfrak{M}_{tf_m, d}^{\tilde{\approx} \bullet}$.

$$t_m : D \times (\mathfrak{M}^s \times \mathfrak{M}^{ab}) \rightarrow D \cup \{\perp\},$$

$$t_m(d, (m^s, m^{ab})) = \begin{cases} d^m, & \text{falls } d \in D_{tf_m}^{\tilde{\approx} \bullet}, (m^s, m^{ab}) \in \mathfrak{M}_{tf_m, d}^{\tilde{\approx} \bullet} \\ \perp, & \text{sonst} \end{cases}$$

Dabei ist

- $D_{tf_m}^{\tilde{\approx} \bullet} := \{d \in D \mid tf_m \tilde{\approx} \bullet d\}$, $D_{tf_m}^{\tilde{\approx} \bullet} \subseteq D$, die Menge aller DMMN-Modelle, auf die tf_m dynamisch kompatibel abgebildet werden kann,
- $d^m \in D$ ein DMMN-Modell, das durch Anwenden des Musters auf d entsteht und die Musterlösung in Form von Annotationen deklariert,
- $\mathfrak{M}_{tf_m, d}^{\tilde{\approx} \bullet} \subseteq \mathfrak{M}^s \times \mathfrak{M}^{ab}$ die Menge aller zu einem bestimmten DMMN-Modell $d \in D_{tf_m}^{\tilde{\approx} \bullet}$ dynamisch kompatiblen Zuordnungen (siehe [Definition 76](#)).

■

Eine Mustertransformation t_m darf dabei grundsätzlich die folgenden Operationen zur Modifikation des eingehenden DMMN-Modells durchführen: (i) Topologieelemente mitsamt deren Typen und Annotationen hinzufügen sowie bereits enthaltene Annotationen modifizieren oder entfernen; (ii) Eigenschaften, Anwendungsartefakte, Schnittstellen, Operationen und Operationsimplementierungen von Topologieelementen, welche nicht existente Komponenten oder Relationen beschreiben, hinzufügen, modifizieren oder entfernen sowie das Topologieelement selbst entfernen. Ein Topologieelement te_i , das eine nicht existente Komponente oder Relation beschreibt, ist folgendermaßen definiert:

$$(State, Uninstalled) \in eigenschaften(te_i)$$

6.3.3 Umsetzung automatisierter Mustertransformationen

Mustertransformationen wurden in den vorangegangenen Abschnitten ausschließlich auf Basis des DMMN-Metamodells beschrieben. In Abhängigkeit der jeweiligen Musterlösung, der Serialisierungsform des DMMN-Modells und der zur Transformation benötigten Konzepte, beispielsweise regelbasierte Transformationen oder imperative Transformationsbeschreibungen [Bie10], eignen sich verschiedene Ansätze zur Implementierung bzw. Modellierung einer Mustertransformation, um diese automatisiert auf DMMN-Modelle anzuwenden.

Während Transformationssprachen wie die *ATLAS Transformation Language (ATL)* [JK05], *Query View Transformation (QVT)* [OMG15] oder *XSL Transformations (XSLT)* [W3C99] explizit zur Beschreibung von Transformationen entwickelt wurden, können auch Programmiersprachen wie Java eingesetzt werden, falls sehr komplexe Transformationslogik implementiert werden muss.

Des Weiteren ist zur automatisierten Transformation von DMMN-Modellen auch der Einsatz von Workflow-Sprachen denkbar, falls eine Transformation beispielsweise semi-automatisiert implementiert werden soll und Nutzerinteraktionen gewünscht oder unumgänglich sind. Über diese Variante können Anwendungsmanager eine Musteranwendung anstoßen, welche während der Transformation von Dritten beeinflusst werden kann, beispielsweise, um aktuelle Unterneh-

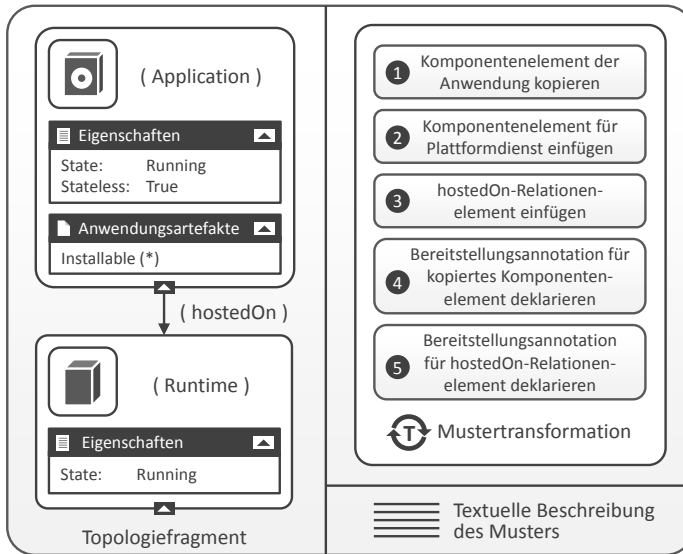


Abbildung 6.4: Beispiel eines automatisierten Managementmusters

mensstrategien, welche noch nicht durch alle Muster erfasst wurden, bei der Anwendung mit einzubeziehen. Ein Überblick verschiedener Transformationsparadigmen, -standards und -technologien wird von Biehl [Bie10] vorgestellt. Die Diskussion weiterer Umsetzungsmöglichkeiten zur automatisierten Transformation von DMMN-Modellen ist somit nicht Gegenstand dieser Arbeit, da diese auf Basis etablierter Technologien realisiert werden können. Im Rahmen der prototypischen Implementierung des PALMA-Frameworks werden Mustertransformationen mittels Java-Programmen umgesetzt, siehe [Abschnitt 8.2.3](#).

6.4 Beispiel eines automatisierten Managementmusters

[Abbildung 6.4](#) erläutert die Bestandteile anhand eines Beispielmusters, das zum Testen der Lauffähigkeit einer Komponente eingesetzt wird. Das abgebildete Muster beschreibt die Bereitstellung einer zustandslosen Anwendungskomponente, welche aktuell auf einer bestimmten Laufzeitumgebung betrieben wird,

auf einem elastischen Plattformdienst, um zu evaluieren, ob die Komponente auf der elastischen Zielumgebung lauffähig ist. Das Muster löst das folgende abstrakte Problem: *Wie kann eine zustandslose, laufende Anwendungskomponente zu Testzwecken elastisch bereitgestellt werden?* Die originale Komponente sowie andere Komponenten des Produktivsystems werden durch die Lösung nicht beeinflusst: Die betroffene Anwendungskomponente wird lediglich mit identischen Artefakten, Schnittstellen und Eigenschaften auf dem elastischen Plattformdienst ausgeliefert. Um dies zu gewährleisten, werden sowohl eingehende als auch ausgehende Relationen der Komponente ignoriert.

Der Kontext dieses Musters, in welchem es angewendet werden kann, wird durch das abgebildete Topologiefragment beschrieben. Das automatisierte Managementmuster kann auf Komponenten des Typs *Application* angewendet werden, sofern diese die *Stateless*-Eigenschaft mit dem positiven Wahrheitswert „True“ erfüllen und über eine *hostedOn*-Relation mit einer Komponente vom Typ *Runtime* verbunden sind, welche ebenfalls aktuell betrieben wird. Des Weiteren muss die Implementierung der Anwendungskomponente in Form eines Anwendungsartefakts verfügbar sein, das zur Bereitstellung der Komponente eingesetzt werden kann. Die Typen der im Fragment repräsentierten Komponenten sind dabei aufgrund der generischen Natur des Musters auf hoher Abstraktionsebene definiert. Dadurch ist das Muster gemäß der Kompatibilitätsrelation $\tilde{\approx}_M$ auf konkretere Typen abbildbar und kann somit zum Umzug unterschiedlicher Anwendungskomponenten eingesetzt werden, sofern deren Anwendungsstruktur kompatibel zum Topologiefragment des Musters ist.

Die Transformation des Musters ist in [Abbildung 6.4](#) auf der rechten Seite durch textuell beschriebene Schritte dargestellt. Dabei wird das betroffene Komponentenelement kopiert und in das DMMN-Modell eingefügt. Da es sich um eine laufende Anwendungskomponente handelt, können die im Instanzmodell repräsentierten Eigenschaften verwendet werden, um die Komponente mit identischer Konfiguration auf der Zielplattform bereitzustellen. Zusätzlich wird ein Komponentenelement für einen Plattformdienst eingefügt, welcher über eine eingehende *hostedOn*-Relation mit der Anwendungskomponente verbunden ist. Anschließend werden an das kopierte Komponentenelement sowie das

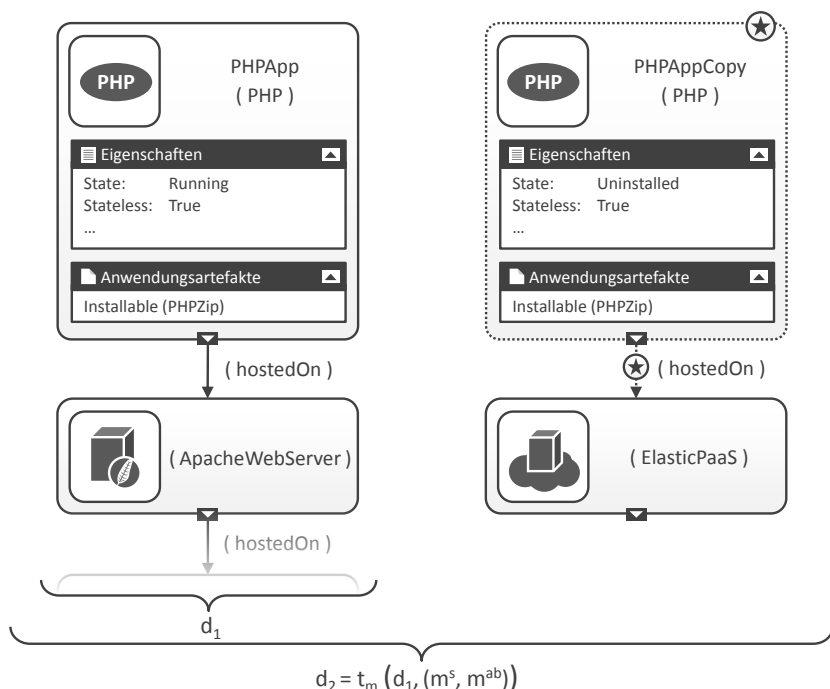


Abbildung 6.5: Eingabe und Resultat der Transformation des Beispielmusters

eingefügte Relationenelement jeweils eine Bereitstellungsannotation deklariert. Für die beiden Annotationen werden dabei keine Konfigurationen spezifiziert, da diese von der Zielplattform abhängig sein können, welche dem Muster aufgrund der abstrakten Musterlösung nicht bekannt ist. [Abbildung 6.5](#) zeigt die Anwendung dieses Musters auf ein Instanzmodell $d_1 \in D$, das als DMN-Modell modelliert ist. Das Instanzmodell beschreibt eine laufende PHP-Anwendung, die auf einem Apache Webserver betrieben wird. Zur Vereinfachung werden weitere Infrastrukturkomponenten, wie das Betriebssystem, sowie Eigenschaften, Artefakte und Schnittstellen der Komponenten und Relationen in dieser Darstellung ausgelassen, da diese keine Relevanz für die beschriebene Transformation haben. Aufgrund der Vererbungshierarchie stellt der Komponententyp *PHP* einen Subtyp von *Application* dar. Gleichermäßen erbt *ApacheWebServer* von *Runtime*.

Gemäß der Kompatibilitätsrelation $\approx_\bullet$ ist das Topologiefragment des Musters zu dieser Anwendungsstruktur kompatibel und kann somit angewendet werden (vgl. Definition 77). Sei $m \in AMM$ das beschriebene automatisierte Managementmuster, dann ist das DMMN-Modell $d_2 \in D$, $d_2 = t_m(d_1, (m^s, m^{ab}))$, das Resultat dessen Anwendung für die einzige dynamisch kompatible Zuordnung zwischen dem Topologiefragment des Musters und dem DMMN-Modell. Dabei enthält das DMMN-Modell d_2 das ursprüngliche Instanzmodell d_1 unverändert als Teilgraphen, da dieses durch die Transformation nicht modifiziert wird.

6.5 Unterschiedliche Abstraktionshöhen automatisierter Muster

Durch eine hohe Abstraktionshöhe von Problem, Kontext und Lösung kann ein automatisiertes Managementmuster generisch auf eine Vielzahl individueller Anwendungen angewendet werden. Beispielsweise kann das im vorherigen Abschnitt vorgestellte Muster zur Replikation beliebiger Anwendungskomponenten eingesetzt werden, unabhängig davon, in welcher Programmiersprache diese implementiert sind oder welche Zielplattform gewählt wird. Aufgrund der abstrakt spezifizierten Lösung erfordert die Anwendung dieses automatisierten Musters jedoch eine nachträgliche, manuelle Verfeinerung des resultierenden DMMN-Modells: Der Plattformdienst muss durch ein konkretes Dienstangebot ersetzt sowie die unverändert kopierten Eigenschaften der Anwendungskomponente angepasst werden. Die PALMA-Methode definiert dazu explizit den Adaptionsschritt 3, in dem das transformierte DMMN-Modell diesbezüglich modifiziert und eine abstrakt deklarierte Lösung hinsichtlich des realen, konkreten Anwendungsfalls verfeinert werden kann. Im Falle einer solchen, in diesem Beispiel notwendigen, manuellen Adaption stellt die Anwendung eines automatisierten Managementmusters einen semi-automatisierten Ansatz dar.

Automatisierte Managementmuster dieses Abstraktionsgrades sind effektiv, um den Lösungsrahmen eines Managementproblems automatisiert vorzugeben, welcher anschließend für individuelle Anwendungsfälle manuell verfeinert werden muss. Häufig tritt ein Problem jedoch für eine Vielzahl ähnlicher Anwendungen auf, die strukturell identisch aufgebaut sind oder identische

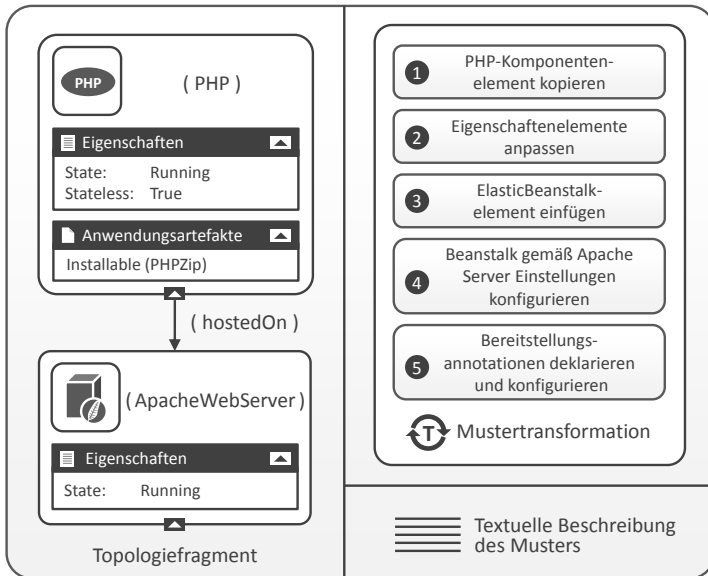


Abbildung 6.6: Beispiel des verfeinerten Managementmusters

Teilstrukturen aufweisen, die vom jeweiligen Managementproblem direkt betroffen sind. Ein Beispiel dafür ist der Umzug aller Anwendungsinstanzen eines bestimmten Typs auf eine konkrete Zielplattform inklusive der Optimierung deren Konfigurationen. Da die manuelle Vorgehensweise für häufig auftretende konkrete Problemfälle zu identischen manuellen Verfeinerungen führt, ist dieses semi-automatisierte Vorgehen in diesen Szenarien nicht effizient. Im Folgenden wird daher eine Anwendung des Konzepts vorgestellt, das den Adaptionsschritt 3 der Lösungsverfeinerung durch Reduktion der Abstraktionshöhe eines automatisierten Managementmusters vollständig automatisiert.

Ausgehend vom abstrakt beschriebenen Beispielmuster des vorherigen Abschnitts zeigt [Abbildung 6.6](#) eine verfeinerte Variante dieses Musters, welches dessen vollautomatisierte Anwendung für einen bestimmten, spezifischen Anwendungsfall ermöglicht: Das verfeinerte Muster konkretisiert sowohl Problem als auch Lösung hinsichtlich PHP-basierter Anwendungen, die auf dem Platt-

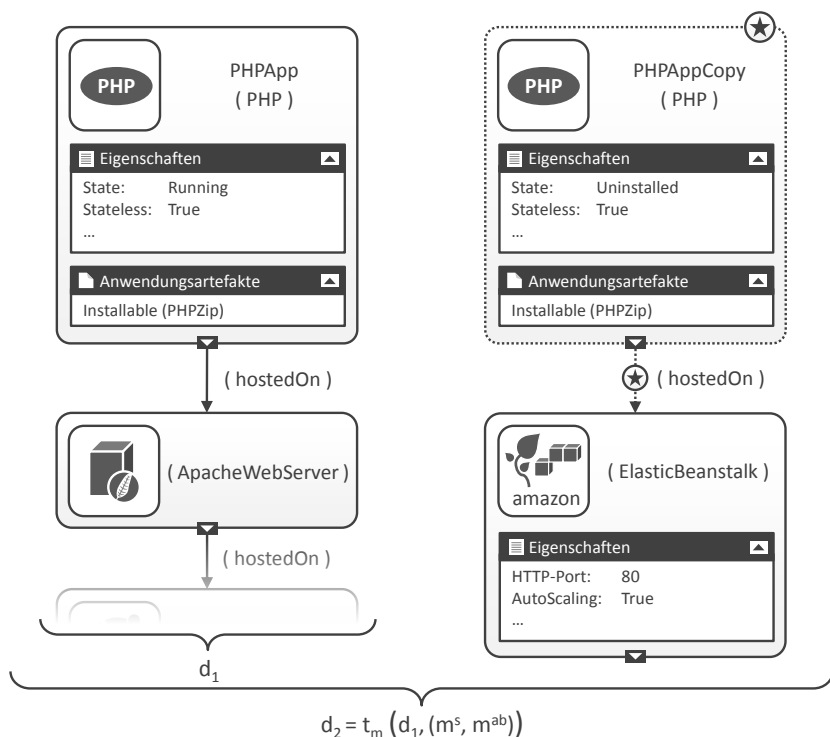


Abbildung 6.7: Eingabe und Resultat der Transformation des verfeinerten automatisierten Managementmusters

formdienst Amazon Beanstalk bereitgestellt werden sollen¹. Die Verfeinerung des Musters für diesen Anwendungsfall resultiert zum einen (i) in einem verfeinerten Topologiefragment, zum anderen (ii) beschreibt die Transformation Schritte, welche für diesen Anwendungsfall spezifisch sind. Das dargestellte Topologiefragment ist somit nur auf PHP-Anwendungen anwendbar, die auf einem Apache Webserver betrieben werden. Die Transformation führt aufgrund der Spezialisierung auf diesen konkreten Anwendungsfall domänenspezifische

¹Technologiespezifische Muster eines sehr niedrigen Abstraktionsgrades, z. B. Muster für Programmiersprachen, werden in einigen Arbeiten auch als *Idiome* bezeichnet, vgl. Buschmann et al. [BMR+96], Coplien [Cop96] und Breitenbücher et al. [BBKL14c].

Schritte durch und kann somit auf die individuellen Charakteristika dieser Spezialisierung eingehen. [Abbildung 6.7](#) zeigt die Anwendung des verfeinerten Musters auf dieselbe Anwendungsinstanz wie beim abstrakten Muster in [Abbildung 6.5](#). Der Unterschied liegt im resultierenden DMMN-Modell, das nach Anwendung des verfeinerten Musters keinerlei weitere manuelle Adaption erfordert und somit vollautomatisiert in das zugehörige imperative Managementmodell transformiert werden kann. Beispielsweise wird der Plattformdienst direkt mit dem konkreten Komponententyp *ElasticBeanstalk* eingefügt. Des Weiteren kann die verfeinerte Transformation Konfigurationen des Webserver und der Anwendung technologiespezifisch interpretieren und die durchzuführende Bereitstellung dementsprechend äquivalent konfigurieren.

Durch diesen Verfeinerungsansatz kann, neben konzeptioneller Managementexpertise wie im vorherigen Beispiel, somit auch bewährtes technologiespezifisches Managementwissen konkreter Probleme erfasst werden, wodurch dieses vollautomatisiert für entsprechende Anwendungsstrukturen ausgeführt werden kann. Die manuelle Verfeinerung resultierender DMMN-Modelle kann dadurch in ihrem Umfang reduziert, im Idealfall vollständig vermieden werden.

6.6 Managementprimitive

Die fundamentalen Bestandteile einer Musterlösung werden auch als *Musterprimitive* (engl. „*pattern primitives*“) [[ZAHD08](#)] bezeichnet, welche beispielsweise in Form von *Architectural Primitives* [[MM03](#)] zur präzisen Beschreibung von IT-Architekturmustern verwendet werden [[ZA05](#); [KA08](#)]. In dieser IT-Domäne repräsentieren Primitive die kleinsten sinnvollen Einheiten auf der Abstraktionsebene der Architektur, z. B. das Konzept des „*Callbacks*“ oder der „*Indirection*“ [[ZA05](#)]. Die Grundannahme des Konzepts besteht darin, dass verschiedene Muster einer Sprache dieselben Primitive zur Beschreibung ihrer Lösungen verwenden, wodurch die Semantik dieser Einheiten musterübergreifend präzise definiert werden kann. Dies unterstützt das Verständnis verschiedener Lösungen innerhalb einer Mustersprache, vereinheitlicht die Terminologie und ermöglicht eine präzise Definition der Semantik fundamentaler Begriffe.

Zur Beschreibung von Transformationen automatisierter Managementmuster werden implizit Primitive eingesetzt, welche in Form von typisierten (i) Komponenten, (ii) Relationen und (iii) Managementannotationen repräsentiert werden, um die betroffenen Entitäten sowie die modellierten Managementaufgaben zu beschreiben. Diese stellen somit *Managementprimitive* zur Beschreibung automatisierter Managementmuster dar. Im Gegensatz zu Architekturprimitiven, deren Semantik in verschiedenen Mustern leicht abweichen kann [ZA05], wird die Semantik von Managementprimitiven in dieser Arbeit musterübergreifend eindeutig definiert: Die Semantik einer Komponente, Relation- oder Managementannotation wird ausschließlich durch deren Typ definiert, welcher, falls definiert, eindeutig ist. Die Verwendung dieser Primitive in einer Lösung eines Managementmusters spiegelt somit stets eine einheitliche Semantik wieder, wodurch die Grundlage zur Übersetzung in korrekte imperative Managementmodelle gelegt wird. Dies spielt eine fundamentale Rolle bei der vollständigen Automatisierung durch technologiespezifische Managementmuster, welche aufgrund der gewünschten fehlenden manuellen Verfeinerung (Schritt 3 der PALMA-Methode) eine direkte Übersetzung der resultierenden deklarativen Managementmodelle in imperative Managementmodelle zur Folge hat und somit keine manuelle Korrektur bzw. Adaption erfordern sollte.

6.7 Automatisierte Muster zur Bereitstellung von Anwendungen

In den vorherigen Abschnitten wurden Muster im Bereich des Anwendungsmanagements automatisiert, die ausgehend vom Instanzmodell einer laufenden Anwendung auszuführende Managementaufgaben deklarieren, um ein deklaratives DMMN-Modell zu erzeugen. Der vorgestellte Ansatz kann neben diesen Managementaspekten gleichermaßen zur Automatisierung von Mustern im Bereich der Anwendungsbereitstellung herangezogen werden. Diese *automatisierten Bereitstellungsmuster* entsprechen in konzeptionellem Aufbau sowie der formalen Form dem Metamodell des automatisierten Managementpendants, werden jedoch zur initialen Bereitstellung ganzer Anwendungen oder Teilen davon eingesetzt und nicht zum Management laufender Anwendungen.

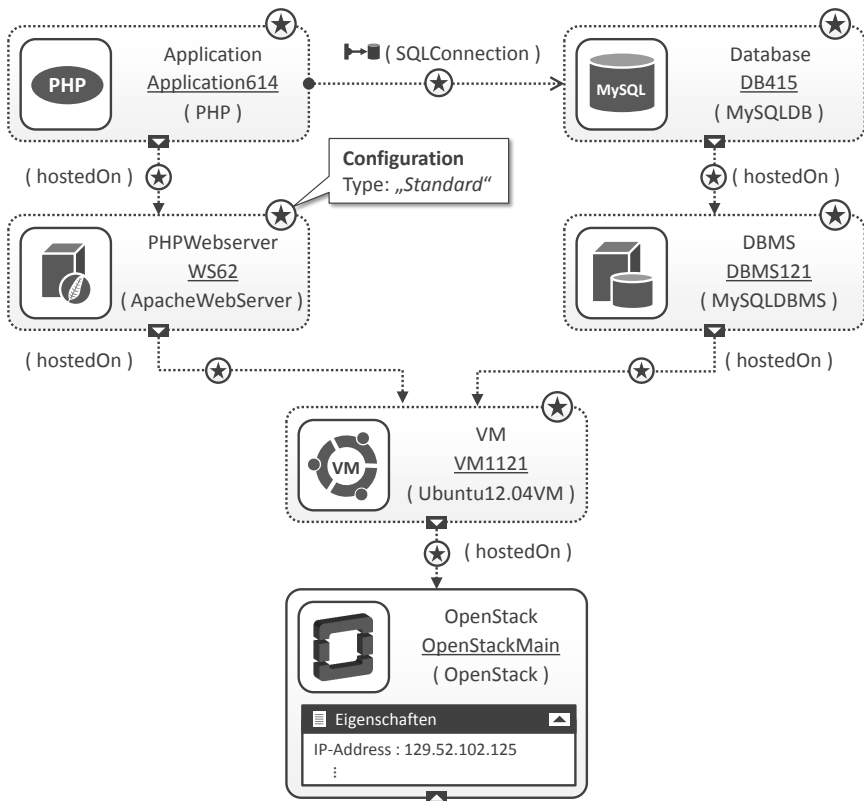


Abbildung 6.8: Generiertes DMMN-Modell des Bereitstellungsmusters

6.7.1 Beispiel eines automatisierten Bereitstellungsmusters

In diesem Abschnitt wird ein Beispiel vorgestellt, das ein (i) abstraktes automatisiertes Bereitstellungsmuster beschreibt, (ii) dessen Verfeinerung hinsichtlich eines spezifischeren Anwendungsfalls diskutiert sowie (iii) die zur Ausführung erforderlichen Management-Planlets und deren Abhängigkeiten beschreibt. Das gewählte Bereitstellungsmuster dient der Modellierung der Bereitstellung einer LAMP-Anwendung¹ auf einer vordefinierten OpenStack-Installation. Das

¹Eine Softwarekombination, in der ein Linux-Betriebssystem eingesetzt wird, um einen Apache Webserver und eine MySQL-Datenbank zur Ausführung einer PHP-Anwendung zu betreiben.

automatisierte Muster definiert für dessen Anwendbarkeit keinerlei Vorbedingungen, d. h. es sind keine Knoten oder Kanten im Topologiefragment des Musters enthalten, wodurch das automatisierte Muster auf beliebige, auch leere, DMMN-Modelle angewendet werden kann. Die Transformation des automatisierten Musters fügt eingehenden DMMN-Modellen die in [Abbildung 6.8](#) simplifiziert dargestellten Komponenten-, Relationen, und Annotationenelemente hinzu. Der Anwendungsmanager muss somit lediglich das gewünschte Anwendungsartefakt für das PHP-Komponentenelement spezifizieren.

Eine Besonderheit stellt dabei die Bereitstellungsannotation dar, welche an den Webserver annotiert wird. Diese spezifiziert eine Konfiguration des Typs *Standard*, wodurch die Semantik dieser Annotation deklarativ auf die Standardinstallation des Webserver konfiguriert wird. Durch ein spezifisches Management-Planlet, welches diese Bereitstellungsannotation für diesen Konfigurationstyp ausführen kann, können so unternehmensweit homogene Konfigurationen LAMP-basierter Anwendungstopologien sichergestellt werden, da diese nicht im DMMN-Modell spezifiziert werden müssen sondern vom Planlet bei der Bereitstellung automatisch umgesetzt und eingehalten werden. Dadurch können spezifische und komplexe Konfigurationen durch abstrakte Konfigurationstypen gebündelt werden, wodurch die Modellierung vereinfacht und die Heterogenität der Anwendungslandschaft reduziert wird. Des Weiteren wird die ausgewählte OpenStack-Installation vom Muster vorgegeben, indem der Endpunkt der Installation sowie Zugangsdaten (nicht dargestellt) direkt im DMMN-Modell spezifiziert werden. Dadurch können unterschiedliche Anwendungsfälle durch individuelle Muster automatisiert werden, etwa die Bereitstellung in einer Testumgebung gegenüber der Bereitstellung in einem Produktivsystem.

Automatisierte Bereitstellungsmuster können, gleichermaßen wie automatisierte Managementmuster, Vorbedingungen definieren, beispielsweise die Existenz einer virtuellen Maschine mit bestimmter Linux-Distribution. Das beschriebene Muster kann somit dahingehend modifiziert werden, dass es nicht die Instanziierung einer neuen virtuellen Maschine auf einer bestimmten, vordefinierten OpenStack-Installation deklariert, sondern eine bereits existierende Maschine verwendet und die Installation der weiteren Komponenten darauf modelliert.

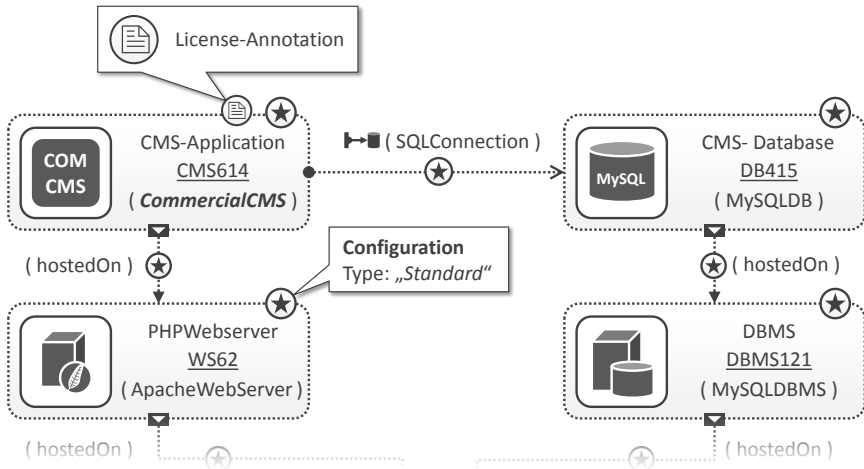


Abbildung 6.9: DMMN-Modell des technologiespezifischen Musters

6.7.2 Verfeinerung des Musters hinsichtlich technologiespezifischer Details

Das im vorherigen Abschnitt vorgestellte Beispiel stellt eine generische, semi-automatisierte Lösung für die Bereitstellung LAMP-basierter Systeme dar, da die resultierenden DMMN-Modelle nachträglich für eine bestimmte Anwendung verfeinert werden müssen. Das Konzept kann jedoch, wie im Folgenden gezeigt, ebenso zur Automatisierung technologiespezifischer Bereitstellungsexpertise eingesetzt werden, vgl. [Abschnitt 6.5](#). Soll beispielsweise die Bereitstellung eines kommerziellen Content Management Systems durch ein automatisiertes Bereitstellungsmuster vordefiniert werden, können anwendungs- bzw. technologiespezifische Details, wie die Verwaltung erforderlicher Lizenzen, in Kombination mit dem Planlet-Ansatz automatisiert werden. [Abbildung 6.9](#) zeigt dazu einen adaptierten Ausschnitt des vorherigen Beispiels, in welchem die automatisierte Transformation des Musters anstatt des generischen PHP-Komponentenelements ein spezifisches Komponentenelement des Typs *CommercialCMS* erzeugt¹. In diesem Beispiel erfordert die Bereitstellung eine dedizierte Lizenz pro Installation

¹Das Beispiel wird unabhängig von einer konkreten CMS-Software geführt, um die generelle Anwendbarkeit des Konzepts zu verdeutlichen.

des CMS, was einem verbreiteten Lizenzmodell unterschiedlichster kommerzieller Software entspricht. Dem DMMN-Modell fügt die Mustertransformation daher eine *Lizenzannotation* hinzu, welche sicherstellt, dass eine Lizenz für diese Komponente vorhanden ist. Die Installation des CMS kann daher erst stattfinden, wenn die Lizenzanforderungen erfüllt sind, d. h. wenn der bereitzustellenden Anwendungsinstanz eine Lizenz zugewiesen bzw. beschafft wurde. Durch Konfiguration der Annotation kann zusätzlich deklarativ spezifiziert werden, in welcher Form die Lizenz beschafft werden soll, beispielsweise ob für jede Bereitstellung eine neue Lizenz erworben oder ob aus vorhandenen Lizenzen ausgewählt werden soll. Die Semantik dieser Annotation sowie möglicher Konfigurationen wird dabei domänenspezifisch definiert, indem dedizierte Planlets entwickelt werden, welche diese Annotationen umsetzen können.

6.7.3 Domänenspezifische Ausführung der Annotationen des Beispiels

Zur Ausführung des generierten DMMN-Modells zur Bereitstellung eines kommerziellen CMS sind spezifische Management-Planlets erforderlich, welche (i) die benötigte Lizenz beschaffen und (ii) diese bei der Installation berücksichtigen. In diesem Abschnitt werden zwei Management-Planlets vorgestellt, welche die Annotationen zur Installation des CMS ausführen können.

Abbildung 6.10 zeigt auf der linken Seite das Topologiefragment eines Planlets mp_1 , welches die Lizenzannotation ausführen kann. Als Vorbedingung spezifiziert das Planlet, dass die CMS-Komponente gegenwärtig nicht installiert sein darf (*State*-Eigenschaft mit dem Wert „Uninstalled“). Der Effekt des Planlets ist die Beschaffung einer Lizenz für das CMS, was durch den modellierten Effekt ausgedrückt wird (*License*-Eigenschaft mit dem Wert „*“). Das Wildcard-Symbol des Sterns spezifiziert hierbei, dass eine Lizenz beschafft wird, deren exakte Ausprägung erst zur Laufzeit bekannt sein wird. Auf der rechten Seite in [Abbildung 6.10](#) ist das Topologiefragment eines Planlets mp_2 abgebildet, welches das CMS auf einem laufenden Apache-Webserver installieren kann. Zur Ausführung der Bereitstellungsannotation definiert das Planlet Vorbedingungen an den Webserver (nicht dargestellt) sowie an das CMS-Komponentenelement,

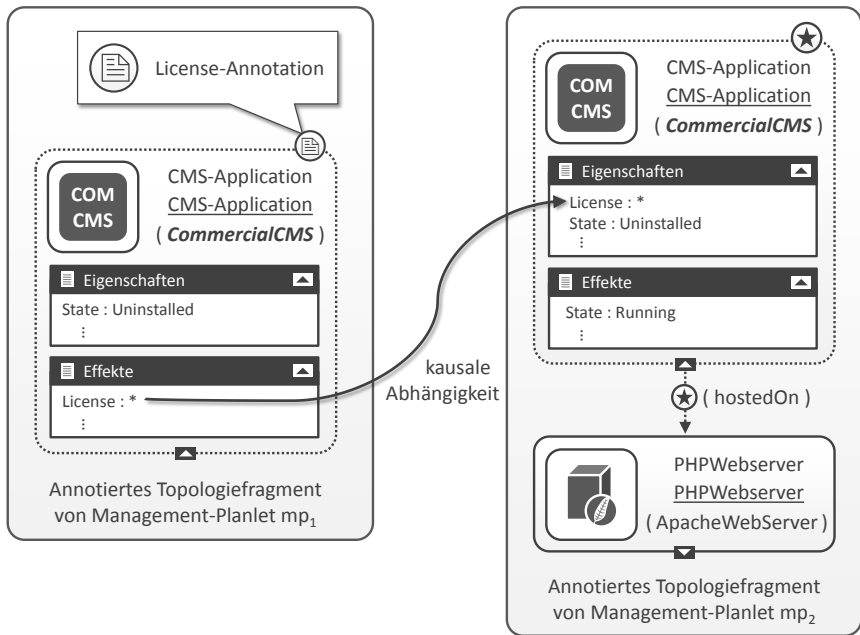


Abbildung 6.10: Planlets zur Ausführung der Annotationen des technologiespezifischen automatisierten Bereitstellungsmusters

insbesondere die Existenz eines beliebigen Werts für die *License*-Eigenschaft, was durch das Wildcard-Symbol modelliert ist. Durch diese Eigenschaft ergibt sich somit eine kausale Abhängigkeit der beiden Planlets, wodurch die Bereitstellung erst nach Beschaffung einer Lizenz ausgeführt werden kann und die Ordnung der beiden Planlets dadurch implizit vorgegeben ist. Die Umsetzung der Management-Planlets, d. h. die Modellierung deren imperativen Managementmodelle, kann dazu beliebige Aktivitäten einbinden, beispielsweise eine manuelle Aktivität (engl. „*Human-Task*“) zum Erwerb einer neuen Lizenz, welche durch einen menschlichen Mitarbeiter ausgeführt wird.

Die Ordnung der Management-Planlets kann vom automatisierten Muster auch explizit mithilfe einer Reihenfolgespezifikation modelliert werden, etwa wenn kein spezieller Lizenzschlüssel erforderlich ist und lediglich die Anzahl verfügbarer Lizenzen eines Unternehmens berücksichtigt werden muss.

6.8 Diskussion und Abgrenzung zu verwandten Arbeiten

Das vorgestellte Konzept ermöglicht es Organisationen, internes Managementwissen auf verschiedenen Abstraktionsebenen generisch, d. h. unabhängig von einzelnen Anwendungen, zu automatisieren. Dadurch kann beispielsweise firmenspezifisches Lizenzmanagement automatisiert und homogen über die gesamte Anwendungslandschaft hinweg umgesetzt werden. Dem kritischen Problem wechselnder Mitarbeiter und dem damit verbundenen Wissensverlust [HZR10] kann somit systematisch durch Automatisierung entgegengewirkt bzw. neues automatisiertes Wissen von Dritten eingekauft werden. Zur systematischen Erstellung automatisierter Managementmuster wurde im Rahmen dieser Dissertation zudem eine Methode [BBL14] publiziert.

Jamshidi et al. [JPCL14] stellen einen Katalog feingranularer Migrationsmuster vor, die bewährtes Managementwissen bezüglich der Migration von Anwendungskomponenten in Cloud-Umgebungen dokumentieren. Die Muster dokumentieren den betroffenen konzeptionellen Architekturausschnitt dabei jeweils vor und nach der Migration, wodurch generische Transitionspläne (engl. „*Migration Transition Graph*“) erstellt werden können, die eine Sequenz verschiedener Muster sowie die jeweils eingehende und resultierende Architektur beschreiben. Diese Dokumentationsform eignet sich besonders zur Automatisierung durch die Ansätze der vorliegenden Arbeit. Insbesondere die konzeptionelle Dokumentation der Ausgangsarchitektur sowie der resultierenden Architektur, welche durch Anwendung des Musters entsteht, unterstützt die Implementierung der Transformation automatisierter Managementmuster.

Die Migration von Anwendungen in Cloud-Umgebungen erfordert enorme Expertise bezüglich Geschäfts- und Technologieaspekten, insbesondere wenn Cloud-Eigenschaften bestmöglich ausgenutzt werden sollen. Anwendungsmigrationsmuster, wie beispielsweise von Jamshidi et al. [JPCL14] vorgestellt, unterstützen dies durch eine konzeptionelle Dokumentation bewährten Wissens. Die Auswahl von geeigneten Mustern und Cloud-Anbietern bleibt jedoch ein manueller Prozess, der bei ungünstigen Entscheidungen weitreichende Folgen haben kann. Chee et al. [CZM+11] stellen dazu einen Ansatz vor, der die Mus-

terauswahl zur Erfüllung spezifizierter Migrationsanforderungen automatisiert. Verfügbare Cloud-Dienste werden dazu hinsichtlich der gebotenen Fähigkeiten (engl. „features“) beschrieben, die betroffene Anwendung hinsichtlich der zum Betrieb benötigten Fähigkeiten. Muster spezifizieren (i) welche Anforderungen sie umsetzen und (ii) welche Fähigkeiten dazu sowohl von Anwendung als auch Zielumgebung erfüllt sein müssen. Basierend auf einer Schnittmengenbildung können dadurch diejenigen Muster und Zielumgebungen berechnet werden, welche die spezifizierten Anforderungen sowie die benötigten Fähigkeiten der Anwendung erfüllen. Im Gegensatz zu diesem Ansatz wird die Auswahl von Managementmustern in der vorliegenden Arbeit nicht gesondert betrachtet und obliegt dem Anwendungsmanager. Teil zukünftiger Forschungsarbeiten ist jedoch anzuwendende Managementmuster optimal und automatisiert zu bestimmen, um ein höherwertiges Geschäftsziel zu erreichen.

Arnold et al. [AEK+07] erfassen bewährtes Wissen über die Bereitstellung von Anwendungskomponenten auf existierenden bzw. geplanten Infrastrukturen formal in Form von Anwendungsstrukturen, die als unvollständige, abstrakt beschriebene Topologiefragmente modelliert werden. Ein solches Topologiefragment wird als Muster bezeichnet und enthält zum einen (i) eine (oder mehrere) bereitzustellende Komponente(n) sowie (ii) deren Anforderungen an andere Komponenten, zu denen die Komponente Abhängigkeiten hat bzw. auf denen sie installiert werden kann. Diese benötigten Komponenten, welche ebenfalls im Topologiefragment enthalten sind, werden als „virtuelle“ Komponenten modelliert und dienen als Schablone, um geeignete Ressourcen zur Bereitstellung der Komponente zu identifizieren. Zudem können diese Muster rekursiv zur Vervollständigung von Zielzustandsmodellen angewendet werden, indem virtuelle Komponenten durch einen Subgraphisomorphismus auf bereits im Modell enthaltene Komponenten, welche die spezifizierten Anforderungen erfüllen, abgebildet werden und die bereitzustellende(n) Komponente(n) sowie deren Relationen zu den enthaltenen Komponenten in das Modell eingefügt werden. In einer späteren Arbeit [AEK+08] stellen die Autoren Algorithmen vor, wie diese Muster automatisiert angewendet und parametrisiert werden können, um Zielkonfigurationen einzuflussender Komponenten und Relationen zu spezifizie-

ren. Diese Vorgehensweise ermöglicht das Einfügen von statischem, strukturellem Bereitstellungs- und Konfigurationswissen in Zielzustandsmodelle in Form neuer Komponenten, Relationen und Einschränkungen. Die vorliegende Arbeit erweitert dieses Konzept, wie in [Abschnitt 6.7](#) für die Anwendungsbereitstellung veranschaulicht, um Transformationen, die existierende Topologiestrukturen beliebig dynamisch verändern können. Dadurch können Anwendungsstrukturen nicht nur durch das Einfügen neuer Entitäten beeinflusst werden, sondern beispielsweise auch im Modell enthaltene Komponenten entfernt oder ausgetauscht werden. Zudem liegt der Hauptfokus der vorliegenden Arbeit auf dem Anwendungsmanagement, weswegen automatisierte Managementmuster über die Bereitstellung hinausgehende Managementaufgaben deklarieren können, z. B. zur effizienten Anwendungsterminierung inklusive eines zuvor ausgeführten Datenexports. Die Überprüfung und Berechnung der Anwendbarkeit automatisierter Managementmuster ist dabei ähnlich zum Ansatz der virtuellen Komponenten, welche über einen Subgraphisomorphismus auf bereits im Modell enthaltene Komponenten abgebildet werden.

Eilam et al. [[EKKP05](#); [EKK+06](#); [EKK09](#)] stellen einen ähnlichen Ansatz vor, um logische Deployment-Modelle, welche ausschließlich die funktionalen Komponenten einer Anwendung, deren Anforderungen an unterliegende Systeme und Relationen zwischen den funktionalen Komponenten beschreiben (engl. „*Logical Deployment Model*“), automatisiert durch eine Transformation in bereitstellbare Topologiemodelle zu verfeinern (engl. „*Physical Deployment Topology*“). Die generierten Topologiemodelle enthalten neben den funktionalen Aspekten zusätzlich alle zum Betrieb nötigen Infrastruktur-, Middleware- und Netzwerkkomponenten, wodurch sie zur automatisierten Bereitstellung der modellierten Anwendung auf einer existierenden Infrastruktur verwendet werden können. Zu dieser automatisierten Verfeinerung werden sogenannte *Transformation Rules* eingesetzt, die feingranulares, bewährtes Wissen über die Bereitstellung von Anwendungskomponenten automatisieren. Ähnlich zu automatisierten Managementmustern bestehen Transformation Rules aus (i) einem Graphen, welcher Vorbedingungen in Form der Anwendungsstruktur spezifiziert auf welche die Regel angewendet werden kann, sowie (ii) einer Transformation, die einen zu

diesem Graphen isomorphen Teil des eingehenden Modells transformiert. Dabei werden beispielsweise Elemente aus dem spezifizierten Graphen hinzugefügt, modifiziert oder gelöscht. Transformation Rules werden solange iterativ angewendet, bis das resultierende Modell eine bereitstellbare Topologie repräsentiert. Das Konzept der automatisierten Managementmuster ist eine Weiterentwicklung der Transformation Rules hinsichtlich des Anwendungsmanagements und unterscheidet sich in folgenden Aspekten: (i) Transformation Rules werden dazu eingesetzt, ein funktionales Anwendungsmodell iterativ in ein bereitstellbares Modell zu verfeinern, wohingegen automatisierte Managementmuster auszuführende Managementaufgaben deklarieren und paradigmatisch über die reine Anwendungsbereitstellung hinausgehen. (ii) Eine Transformation Rule verfeinert lediglich das eingehende Modell hinsichtlich dessen Struktur. Automatisierte Managementmuster hingegen können neben diesen zustandsmodifizierenden Managementaufgaben auch zustandserhaltende Aufgaben sowie deren Ausführungsreihenfolge spezifizieren, was zur Umsetzung komplexer Managementprozesse gemäß der in [Abschnitt 3.2](#) dokumentierten Anforderungen eine grundlegende Voraussetzung darstellt. (iii) Vorbedingungen von Transformation Rules spezifizieren ausschließlich die Struktur sowie Eigenschaften der Komponenten, auf welche die Regel zur Verfeinerung der modellierten Struktur angewendet werden kann. Im Gegensatz dazu können annotierte Topologiefragmente managementrelevante Vorbedingungen, wie existierende Anwendungsartefakte, Managementschnittstellen oder bereits deklarierte Managementaufgaben in Form von Annotationen, spezifizieren.

Die Anwendbarkeit automatisierter Managementmuster wird bestimmt, indem dessen Topologiefragment mit der Struktur der Anwendung auf Kompatibilität überprüft wird. Haitzer und Zdun [HZ15] nutzen einen ähnlichen Vergleichsansatz, jedoch mit dem Ziel, Muster in Softwarecode zu entdecken. Eden et al. [EYG97] automatisieren die Anwendung von Entwurfsmustern (engl. „*Design Patterns*“) im Bereich der Anwendungsentwicklung ähnlich zu automatisierten Managementmustern, indem die Musterlösung als generische Transformation des Quellcodes von Anwendungen implementiert wird.

KAPITEL 7

KONTEXTSENSITIVE ANALYSE DEKLARATIVER MANAGEMENTAUFGABEN

Die Automatisierung von Managementmustern in [Kapitel 6](#) unterstützt den Anwendungsmanager systematisch bei der Deklaration von Managementzielen. Die Durchführung komplexer, umfangreicher Managementaufgaben erfordert jedoch häufig eine manuelle Bearbeitung des resultierenden DMMN-Modells, wodurch Managementfehler entstehen und Optimierungspotenziale durch Unkenntnis oder Unachtsamkeit übersehen werden können.

Um eine erfolgreiche Managementdurchführung zu gewährleisten, stellt dieser Abschnitt das Konzept der *automatisierten Analysemuster* vor, um deklarative Managementaufgaben kontextsensitiv auf Fehler und Optimierungspotenziale zu analysieren und diese gegebenenfalls automatisiert zu beheben bzw. auszuschöpfen. Der Ansatz automatisiert den dritten Schritt der PALMA-Methode (siehe [Kapitel 3](#)) und stellt [Forschungsbeitrag 5](#) der vorliegenden Arbeit dar.

7.1 Motivation

Die Automatisierung von Managementmustern unterstützt den Anwendungsmanager bei der deklarativen Modellierung von Managementaufgaben, indem bewährte Managementexpertise automatisiert für individuelle Anwendungen ausgeführt werden kann. In Kombination mit der automatisierten Plangenerierung werden manuelle Fehlerquellen systematisch eliminiert und eine erhöhte Effizienz der Managementdurchführung gewährleistet.

Das Management komplexer Anwendungen bzw. die Modellierung holistischer, umfangreicher Managementaufgaben erfordert jedoch häufig die kombinierte Anwendung mehrerer Muster sowie eine manuelle Bearbeitung des resultierenden DMMN-Modells durch den Anwendungsmanager. Beispielsweise, um anwendungsspezifische Details zu berücksichtigen, welche sich aufgrund ihrer Seltenheit nicht effizient in Form von automatisierten Managementmustern umsetzen lassen. Durch die manuelle Adaption eines DMMN-Modells können jedoch sowohl auf konzeptioneller als auch auf technischer Ebene Managementfehler begangen werden sowie Optimierungspotenziale hinsichtlich der Umsetzung beabsichtigter Managementziele übersehen werden. Zum Beispiel kann die fehlende Installation eines Treibers auf einem Webserver dazu führen, dass eine Anwendung nicht auf eine neue Datenbank zugreifen kann, deren Bereitstellung vom Anwendungsmanager gleichermaßen wie die Außerbetriebnahme der ursprünglichen Datenbank eines anderen Typs modelliert wurde. Dieser scheinbar offensichtliche und leicht zu behebbende Fehler kann jedoch durch Unkenntnis bzw. falschen Annahmen des Anwendungsmanagers ungewollt auftreten. Webserver haben beispielsweise häufig einen Treiber für MySQL-Datenbanken installiert, nicht jedoch für andere SQL-Datenbanken. Wird das skizzierte DMMN-Modell ausgeführt, kann die Anwendung folglich nicht zur neuen Datenbank verbinden und ist somit nicht mehr funktionsfähig. Neben solch drastischen Managementfehlern können Managementtechnologien, welche einer hohen Änderungsrate und Heterogenität hinsichtlich deren Schnittstellen und Funktionen unterliegen, schnell zu ähnlichen Problemen führen. Cloud-Anbieter wie Amazon und Google setzen beispielsweise unter-

schiedliche Sicherheitsmechanismen ein, welche den Zugriff auf bereitgestellte virtuelle Maschinen regeln. Modelliert ein DMMN-Modell die Bereitstellung einer Anwendung in der Amazon-Cloud, die auf mehrere virtuelle Maschinen verteilt ist, kann der manuelle Austausch des Cloud-Anbieters im DMMN-Modell zu Problemen führen. Beispielsweise, wenn dieselbe Bereitstellung aufgrund unterschiedlicher Sicherheitsmechanismen nicht in der Cloud von Google funktioniert, da die virtuellen Maschinen zur Kommunikation eine andere Konfiguration erfordern. Ohne detaillierte Kenntnisse dieser heterogenen Managementmechanismen kann eine erfolgreiche Managementdurchführung somit nicht vom Anwendungsmanager gewährleistet werden (siehe [Abschnitt 2.2.3.1](#)).

Der Übergang von Managementfehler zu Optimierungspotenzial ist dabei fließend. Wird eine Web-Anwendung von einem Anbieter zu einem anderen migriert, stellt die Aktualisierung der zugehörigen Internetdomain den Anwendungsmanager vor eine nicht-triviale Herausforderung: Wird die alte Installation außer Betrieb genommen bevor alle DNS-Server die neue Installation referenzieren, kann die Verfügbarkeit der Anwendung für den Benutzer in Abhängigkeit der Lokalität negativ beeinflusst werden. Da sich dieses Problem nach geraumer Zeit selbst löst, kann das Abwarten bezüglich der Außerbetriebnahme der alten Installation als Optimierungspotenzial verstanden werden.

Die Ursachen für diese Art Fehler bzw. Optimierungspotenziale resultieren zum einen aus der enormen Komplexität des Anwendungsmanagements, wodurch Unkenntnis eine der Hauptursachen von Fehlern darstellt - insbesondere wenn dem Anwendungsmanager kritische Abhängigkeiten zwischen Komponenten unbekannt sind. Zum anderen können Unachtsamkeiten bzw. ein unüberschaubares DMMN-Modell zu ungewollten Effekten manuell deklarerter Managementannotationen führen, was aus der hohen Fehlerrate menschlichen Handels resultiert (vgl. [Abschnitt 2.2.3.1](#)). Die manuelle Adaption von DMMN-Modellen durch den Anwendungsmanager gefährdet somit die erfolgreiche Managementdurchführung. In diesem Kapitel wird ein musterbasiertes Verfahren vorgestellt, um Managementfehler und Optimierungspotenziale in DMMN-Modellen automatisiert zu erkennen sowie zu beheben bzw. auszuschöpfen.

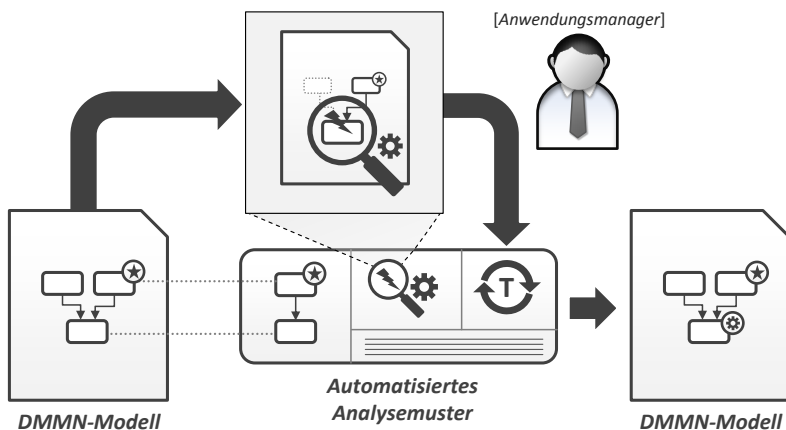


Abbildung 7.1: Grundkonzept der automatisierten Analysemuster

7.2 Idee und Grundkonzept

Im Gegensatz zu vielen existierenden Managementtechnologien ermöglicht DMMN die kontextsensitive Analyse der deklarierten Managementaufgaben, da sowohl die Anwendungsstruktur, deren Zustand sowie alle auszuführenden Aufgaben in DMMN-Modellen beschrieben sind (siehe [Abschnitt 2.3.7](#)). Die zentrale Idee dieses Forschungsbeitrags ist daher die automatisierte Erkennung und Behebung von Managementfehlern, welche die erfolgreiche Managementdurchführung gefährden oder verhindern, sowie die Optimierung der Managementdurchführung durch Analyse des Managementkontextes, in welchem die Managementaufgaben ausgeführt werden. Dabei steht die Behandlung von Managementfehlern im Fokus, welche ohne detaillierte fachspezifische Expertise bezüglich der auszuführenden Managementaufgaben nicht offensichtlich für den Anwendungsmanager erkennbar sind. In diesem Kapitel wird dazu das Konzept der automatisierten Analysemuster eingeführt, um Managementannotationen in DMMN-Modellen automatisiert kontextsensitiv zu (i) analysieren, (ii) eventuelle Fehler und Optimierungsmöglichkeiten zu erkennen und diese (iii) durch Transformation des DMMN-Modells zu beheben bzw. auszuschöpfen. [Abbildung 7.1](#) zeigt eine grafische Darstellung des Konzepts.

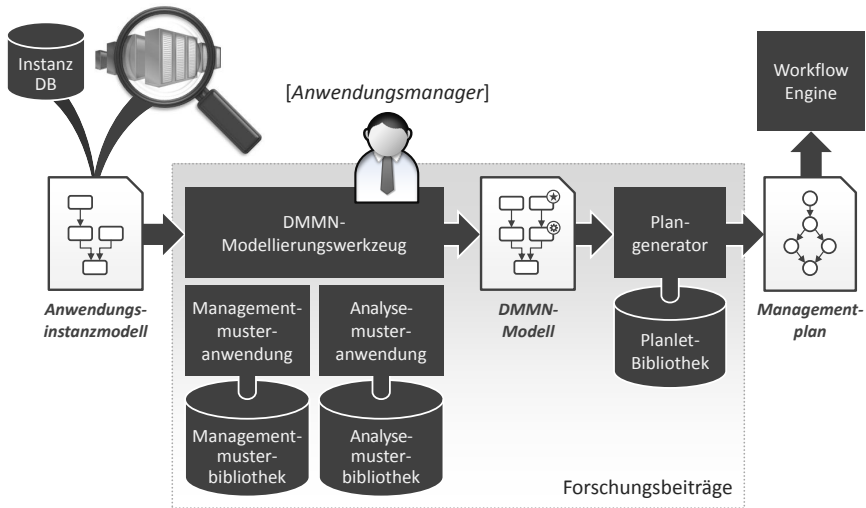


Abbildung 7.2: Erweiterung des Frameworks um automatisierte Analysemuster

In [Abbildung 7.2](#) ist dieser Ansatz im Rahmen des PALMA-Frameworks dargestellt. Während die vorhergehenden Forschungsbeiträge die Modellierung von DMMN-Modellen ([Kapitel 4](#)), deren vollautomatische Transformation in imperative Managementmodelle ([Kapitel 5](#)) sowie die automatisierte Anwendung von Managementmustern ermöglichen ([Kapitel 6](#)), erweitert dieser Forschungsbeitrag das Modellierungswerkzeug des PALMA-Frameworks um die automatisierte Erkennung und Behebung von Managementfehlern und Optimierungspotenzialen in DMMN-Modellen durch automatisierte Analysemuster. Durch den Forschungsbeitrag werden Anwendungsmanager bei der Modellierung korrekter und optimierter DMMN-Modelle unterstützt, indem automatisierte Analysemuster selbständig vom Modellierungswerkzeug zur Erkennung von Fehlern evaluiert werden und im positiven Fall eine Rückmeldung an den Modellierenden liefern. Dieser kann die automatisierte Transformation des Musters dann, falls gewünscht, zur Behebung des erkannten Fehlers bzw. zur Ausschöpfung des Optimierungspotenzials ausführen lassen. Die Architektur und prototypische Realisierung des Konzepts wird detailliert in [Kapitel 8](#) vorgestellt.

7.3 Automatisierte Analysemuster

Dieser Abschnitt führt das Konzept der automatisierten Analysemuster ein, welche die Erkennung von Managementfehlern und Optimierungspotenzialen in DMMN-Modellen sowie deren Behebung bzw. Ausschöpfung ermöglichen.

7.3.1 Kontextsensitive Analyse deklarativer Managementaufgaben

Im Gegensatz zu vielen Managementmodellen, wie beispielsweise Managementplänen (vgl. [Abschnitt 2.3.7](#)), beschreiben DMMN-Modelle neben den eigentlichen Managementaufgaben auch den Kontext, in welchem sie ausgeführt werden, in Form von Anwendungsstruktur und Instanzinformationen (vgl. [Definition 11](#)). Dadurch kann eine Managementannotation, welche für ein Topologieelement deklariert ist, *kontextsensitiv* analysiert werden, da die umgebenen Komponenten und angrenzenden Relationen durch das DMMN-Modell bekannt sind. Soll beispielsweise eine Komponente außer Betrieb genommen werden, welche noch von anderen Komponenten verwendet wird, sind diese Relationen explizit im DMMN-Modell enthalten und ermöglichen die Erkennung dieses Managementfehlers durch Analyse des Kontextes, in welchem die Außerbetriebnahme ausgeführt wird. Ein imperativer Managementplan würde lediglich die Managementaufgabe modellieren, wodurch die oben beschriebenen Probleme bezüglich noch existierender Relationen nicht durch Analyse des Prozessmodells erkannt werden können. Daher definiert die PALMA-Methode einen expliziten Analyseschritt, um derart Managementfehler in deklarativen DMMN-Modellen vor der Transformation in imperative Modelle zu erkennen.

7.3.2 Automatisierung der kontextsensitiven Analyse

Diese Kontextinformationen werden von automatisierten Analysemustern zur Erkennung von Fehlern und Optimierungspotenzialen verwendet. Automatisierte Analysemuster sind eine Erweiterung des Konzepts der automatisierten Managementmuster: Eine formalisierte Anwendungsstruktur definiert den Kontext des Musters, in welchem es einen Managementfehler bzw. ein Optimie-

rungspotenzial erkennen kann, während eine Transformation die Musterlösung automatisiert. Der Unterschied liegt im Umgang mit dem Problem, welches durch das Muster gelöst wird. Automatisierte Managementmuster werden manuell vom Anwendungsmanager ausgewählt, nachdem dieser ein zu lösendes Managementproblem erkannt hat, beispielsweise, dass eine Anwendungskomponente migriert werden muss. Das Erkennen des Managementproblems liegt folglich im Verantwortungsbereich des Anwendungsmanagers und wird somit für Managementmuster nicht automatisiert. Automatisierte Analysemuster hingegen werden dazu eingesetzt, ein Problem in Form eines Managementfehlers oder Optimierungspotenzials innerhalb eines deklarativen Managementmodells sowohl automatisiert zu lösen als auch dieses automatisch zu erkennen. Die Überprüfung, ob der definierte Kontext des Musters durch ein bestimmtes deklaratives Managementmodell gegeben ist, reicht somit für die Automatisierung von Analysemustern nicht aus, da diese im Regelfall dem Anwendungsmanager unbekannte Probleme lösen und die Problemerkennung somit ebenfalls automatisiert werden muss. Im Gegensatz zur Automatisierung von Managementmustern definiert ein automatisiertes Analysemuster daher zudem einen *Analysealgorithmus*, welcher auf den formal definierten Kontext angewendet werden kann, um das eventuell darin enthaltene Problem, welches durch das Muster gelöst wird, zu erkennen. Durch Kombination dieser drei Konzepte können Analysemuster vollautomatisiert auf deklarativen Managementmodellen evaluiert und zur Behebung des jeweiligen Problems eingesetzt werden.

Definition 79 (Automatisiertes Analysemuster - informell). Ein *automatisiertes Analysemuster* ist eine ausführbare Transformation, die das Problem eines Managementfehlers bzw. Optimierungspotenzials in einem deklarativen Managementmodell generisch löst und deren Anwendbarkeit durch eine formale Spezifikation in Kombination mit einem Analysealgorithmus definiert ist. ■

Gleichermaßen wie Managementmuster können Analysemuster auf unterschiedlichen Abstraktionsebenen definiert und automatisiert werden.

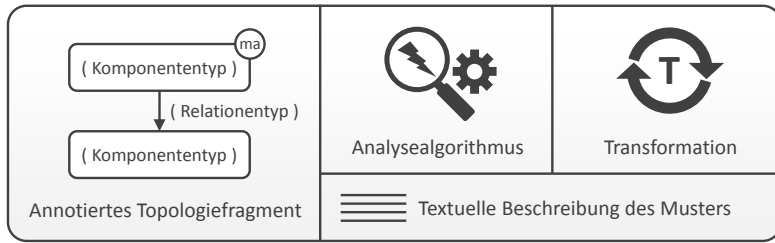


Abbildung 7.3: Aufbau eines automatisierten Analysemodells

7.3.3 Konzeptioneller Aufbau eines automatisierten Analysemodells

Ein automatisiertes Analysemodell besteht konzeptionell aus vier Bestandteilen: (i) Spezifikation des Modulkontextes, (ii) Transformation, welche die Musterlösung automatisiert ausführt, (iii) Analysealgorithmus zur Erkennung des Problems sowie (iv) einer textuellen Beschreibung des Analysemodells.

Die Spezifikation des Modulkontextes definiert wie bei automatisierten Managementmodellen die Anwendungsstruktur, auf die das Modell angewendet werden kann, formal in Form eines DMMN-Modells. Dieses wird ebenfalls als *annotiertes Topologiefragment* bezeichnet. Anders als bei Planlets definieren darin deklarierte Annotationen die Managementaufgaben, bezüglich denen das Modell eventuelle Managementfehler bzw. Optimierungspotenziale erkennt.

Zur Erkennung des Problems innerhalb des formal definierten Modulkontextes implementiert der zweite Bestandteil einen *Analysealgorithmus*, welcher die Logik zur Erkennung des jeweiligen Fehlers bzw. Optimierungspotenzials automatisiert. Der Algorithmus konsumiert das betroffene DMMN-Modell und analysiert die darin enthaltenen Elemente auf das vom Modell lösbare Problem.

Der dritte Bestandteil stellt die *Transformation* dar, welche das betroffene DMMN-Modell bei erkanntem Problem gemäß der Musterlösung transformiert.

Der vierte Bestandteil eines automatisierten Analysemodells stellt die textuelle Beschreibung des Modells dar, die im typischen Modellformat dokumentiert ist.

7.3.4 Formales Metamodell automatisierter Analysemuster

In diesem Abschnitt wird das formale Metamodell für automatisierte Analysemuster vorgestellt, das die Grundlage der folgenden Abschnitte darstellt.

Definition 80 (Automatisiertes Analysemuster). Die Menge aller automatisierten Analysemuster ist definiert als AAM . Sei $aam \in AAM$ ein automatisiertes Analysemuster, dann ist aam definiert als Quadrupel:

$$AAM \subseteq D \times AA \times T \times \Sigma^+$$

$$aam = (tf_{aam}, aa_{aam}, t_{aam}, b_{aam})$$

Dabei ist

- $tf_{aam} \in D$ das Topologiefragment, das den Kontext, in dem das Analysemuster angewendet werden kann, in Form eines DMMN-Modells spezifiziert,
- $AA := \{aa \mid aa : D \times (\mathfrak{M}^s \times \mathfrak{M}^{ab}) \rightarrow \mathbb{B} \cup \{\perp\}\}$ die Menge aller Analysealgorithmen aller automatisierten Analysemuster,
- $aa_{aam} \in AA$ eine Abbildung, welche das Problem des Musters in Form eines automatisierten Analysealgorithmus erkennt. Der Algorithmus bildet DMMN-Modelle $\{d_1, \dots, d_n\} \in D$, auf die tf_{aam} durch eine dynamisch kompatible Zuordnung $(m^s, m^{ab}) \in \mathfrak{M}^s \times \mathfrak{M}^{ab}$ abgebildet werden kann, auf einen Wahrheitswert ab, der besagt, ob das Problem bezüglich der durch (m^s, m^{ab}) zugeordneten Topologieelemente auftritt (siehe [Abschnitt 7.3.5](#)).
- $T := \{t \mid t : D \times (\mathfrak{M}^s \times \mathfrak{M}^{ab}) \rightarrow D \cup \{\perp\}\}$ die Menge aller Transformationen aller automatisierten Muster,
- $t_{aam} \in T$ eine Abbildung, welche die Lösung des Analyseusters als automatisierte Transformation beschreibt (Definition und Realisierung identisch zu Managementmustern, vgl. [Abschnitt 6.3.2](#)).
- b_{aam} die nicht-leere textuelle Beschreibung des Musters. ■

7.3.5 Anwendbarkeit automatisierter Analysemuster

Ein automatisiertes Analysemuster kann auf ein DMMN-Modell $d \in D$ genau dann angewendet werden, wenn der Kontext des Musters durch d gegeben ist und darin das Problem bzw. Optimierungspotenzial erkannt wird, welches vom Muster gelöst wird. Ähnlich wie bei automatisierten Managementmustern werden zur Überprüfung der Anwendbarkeit dynamisch kompatible Zuordnungen eingesetzt, die mit dem Analysealgorithmus des zu evaluierenden Analyseusters kombiniert werden. Die Anwendbarkeit eines automatisierten Analyseusters wird folglich als formale Kompatibilitätsrelation definiert:

Definition 81 (Kompatibilität von Analysemustern und DMMN-Modellen). Sei $aam \in AAM$ ein automatisiertes Analysemuster, tf_{aam} dessen Topologiefragment, aa_{aam} dessen Analysealgorithmus und $d \in D$ ein DMMN-Modell, dann wird die Kompatibilitätsrelation $\tilde{\approx}_{AAM}$ zur Bestimmung der Anwendbarkeit von aam auf das DMMN-Modell d wie folgt formal definiert:

$$\begin{aligned} \tilde{\approx}_{AAM} &\subseteq AAM \times D \\ aam \tilde{\approx}_{AAM} d &:\Leftrightarrow \exists m^s \in \mathcal{M}^s \exists m^{ab} \in \mathcal{M}^{ab} : kompatibel^*(tf_{aam}, d, m^s, m^{ab}) \wedge \\ &\quad aa_{aam}(d, (m^s, m^{ab})) = \text{true} \end{aligned}$$

■

Ein automatisiertes Analysemuster aam hat somit einen Fehler bzw. ein Optimierungspotential in d erkannt, wenn gilt: $aam \tilde{\approx}_{AAM} d$. Die betroffenen Managementaufgaben werden durch die jeweilige dynamisch kompatible Zuordnung spezifiziert, welche durch [Algorithmus 6.1](#) berechnet werden können. Stehen Muster und DMMN-Modell dementsprechend in Relation zueinander, kann das Muster das DMMN-Modell automatisiert mittels t_{aam} transformieren, um das erkannte Problem zu lösen. Die in dieser Definition verwendeten Analysealgorithmen werden als Abbildungen formalisiert, die für alle automatisierten Analysemuster $aam \in AAM$ folgendermaßen definiert sind:

Definition 82 (Analysealgorithmus). Sei $aam \in AAM$ ein automatisiertes Analysemuster, dann wird dessen Analysealgorithmus $aa_{aam} = \pi_2(aam)$ zur Erkennung eines Managementfehlers bzw. Optimierungspotenzials in d bezüglich der im Kontext von tf_{aam} spezifizierten Managementannotationen als Abbildung realisiert, welche für einen eingeschränkten Definitionsbereich einen Wahrheitswert liefert. Dieser Definitionsbereich ergibt sich aus $D_{tf_{aam}}^{\tilde{\bullet}}$ und $\mathfrak{M}_{tf_{aam},d}^{\tilde{\bullet}}$.

$$aa_{aam} : D \times (\mathfrak{M}^s \times \mathfrak{M}^{ab}) \rightarrow \mathbb{B} \cup \{\perp\}, aa_{aam} \in AA,$$

$$aa_{aam}(d, (m^s, m^{ab})) = \begin{cases} \text{true,} & \text{falls } d \in D_{tf_{aam}}^{\tilde{\bullet}}, (m^s, m^{ab}) \in \mathfrak{M}_{tf_{aam},d}^{\tilde{\bullet}} \\ & \text{und das Problem festgestellt wurde} \\ \text{false,} & \text{falls } d \in D_{tf_{aam}}^{\tilde{\bullet}}, (m^s, m^{ab}) \in \mathfrak{M}_{tf_{aam},d}^{\tilde{\bullet}} \\ & \text{und das Problem nicht festgestellt wurde} \\ \perp, & \text{sonst} \end{cases} \quad \blacksquare$$

Ein automatisiertes Analysemuster aam erkennt Probleme bzw. Optimierungspotenziale in d somit jeweils bezüglich einer dynamisch kompatiblen Zuordnung $(m^s, m^{ab}) \in \mathfrak{M}_{tf_{aam},d}^{\tilde{\bullet}}$, für deren zugeordnete Topologieelemente der Analysealgorithmus einen positiven Wahrheitswert als Ergebnis zurückliefert. Die Menge all dieser *problemidentifizierenden Zuordnungen* wird definiert als:

Definition 83 (Menge aller problemidentifizierenden Zuordnungen). Sei $aam \in AAM$ ein automatisiertes Analysemuster, tf_{aam} dessen Topologiefragment und $d \in D$ ein DMMN-Modell, dann enthält die Menge $\mathfrak{M}_{tf_{aam},d}^{\tilde{\bullet},\otimes}$ alle problemidentifizierenden Zuordnungen zwischen tf_{aam} und d , für die das automatisierte Analysemuster das jeweilige Problem in d erkannt hat:

$$\mathfrak{M}_{tf_{aam},d}^{\tilde{\bullet},\otimes} := \left\{ (m^s, m^{ab}) \in \mathfrak{M}^s \times \mathfrak{M}^{ab} \mid (tf_{aam}, d, m^s, m^{ab}) \in \text{kompatibel}^\bullet \wedge aa_{aam}(d, (m^s, m^{ab})) = \text{true} \right\} \quad \blacksquare$$

7.3.6 Algorithmus zur Berechnung problemidentifizierender Zuordnungen

Die Menge aller problemidentifizierenden Zuordnungen zwischen einem automatisierten Analysemuster und einem DMMN-Modell definiert somit die Anwendbarkeit des Musters, d. h. jede dieser Zuordnungen zeigt auf den Teil im DMMN-Modell, in welchem das vom Muster gelöste Problem enthalten ist. Die Menge kann wie folgt durch [Algorithmus 7.1](#) berechnet werden:

Algorithmus 7.1 BerechneKompatibleZuordnungen $\tilde{\cdot}^{\otimes}(aam, d)$

```
1:  $\mathcal{M}_{tf_{aam}, d}^{\tilde{\cdot}^{\otimes}} := \emptyset$ 
2: // Berechne alle dynamisch kompatiblen Zuordnungen zwischen  $tf_{aam}$  und  $d$ 
3:  $\mathcal{M}_{tf_{aam}, d}^{\tilde{\cdot}} := \text{BerechneKompatibleZuordnungen}^{\tilde{\cdot}}(tf_{aam}, d)$ 
4: // Das Muster ist für jede dynamisch kompatible Zuordnung in  $\mathcal{M}_{tf_{aam}, d}^{\tilde{\cdot}}$  anwendbar;
5: // für die der Analysealgorithmus das Problem bzw. das Optimierungspotenzial erkennt
6: for all  $((m^s, m^{ab}) \in \mathcal{M}_{tf_{aam}, d}^{\tilde{\cdot}})$  do
7:   if  $(aa_{aam}(d, (m^s, m^{ab})) = \text{true})$  then
8:      $\mathcal{M}_{tf_{aam}, d}^{\tilde{\cdot}^{\otimes}} := \mathcal{M}_{tf_{aam}, d}^{\tilde{\cdot}^{\otimes}} \cup \{(m^s, m^{ab})\}$ 
9:   end if
10: end for
11: return  $\mathcal{M}_{tf_{aam}, d}^{\tilde{\cdot}^{\otimes}}$ 
```

Der Algorithmus berechnet zunächst die Menge aller dynamisch kompatiblen Zuordnungen zwischen dem Topologiefragment des Musters tf_{aam} und dem DMMN-Modell d (Zeile 3). Für jede dieser Zuordnungen wird anschließend überprüft, ob der Analysealgorithmus des Musters bezüglich des zugeordneten Subgraphen im DMMN-Modell das vom Analysemuster gelöste Problem erkennt (Zeilen 6–10). Im positiven Fall wird die Zuordnung in die resultierende Menge aufgenommen (Zeile 11). Mithilfe dieses Algorithmus kann das Modellierungswerkzeug DMMN-Modelle auf mögliche Managementfehler und Optimierungspotenziale untersuchen, welche anschließend, falls vom Anwendungsmanager gewünscht, durch Ausführung der Mustertransformation gelöst werden können. Die Implementierung wird in [Abschnitt 8.2](#) vorgestellt.

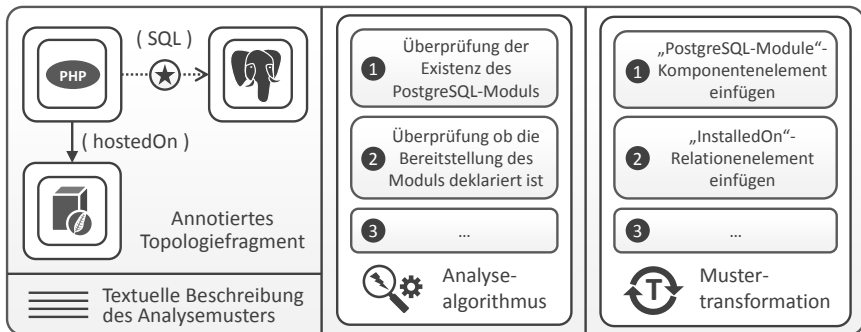


Abbildung 7.4: Beispiel eines automatisierten Analysemodells

7.3.7 Beispiel eines automatisierten Analysemodells

Abbildung 7.4 zeigt ein Beispiel eines automatisierten Analysemodells auf technischer Abstraktionsebene. Das gezeigte Analysemodell erkennt und löst das eingangs erwähnte Problem eines fehlenden Treibers seitens eines Web-servers, um zu einer bestimmten Typ Datenbank zu verbinden. In diesem Beispiel handelt es sich um einen Apache Webserver, der zum Betrieb einer PHP-Anwendung eingesetzt wird. Diese soll zu einer PostgreSQL-Datenbank verbinden. Diese Konstellation ist im Topologiefragment des abgebildeten Modells modelliert und definiert dadurch den Kontext des Modells: Eine Relation des Typs *SQL-Connection*¹ soll zwischen einer Komponente des Typs *PHP* zu einer Komponente des Typs *PostgreSQLDB* instantiiert werden, was durch eine Bereitstellungsannotation an der Relation deklarativ modelliert ist. Tritt innerhalb eines DMMN-Modells dieses Topologiefragment als Teilgraph auf, ist der Kontext des Modells gegeben und es kann das Modell auf den eventuell fehlenden Treiber mittels des skizzierten Analysealgorithmus untersuchen. Ist der Treiber tatsächlich (i) nicht bereits installiert oder (ii) dessen Bereitstellung nicht modelliert, kann dies durch Ausführung der Transformation behoben werden, welche den fehlenden Treiber ins Modell einfügt und dessen Installation deklariert. Zu beachten ist bei diesem Beispiel, dass das Topologiefragment

¹Aus Platzgründen werden Komponentenelemente in Abbildung 7.4 nur durch Icons beschrieben

ausschließlich die Bereitstellung der SQL-Verbindung als Vorbedingung spezifiziert. Ob die anderen im Topologiefragment repräsentierten Komponenten und Relationen erst durch das zu untersuchende DMMN-Modell bereitgestellt werden, ob diese schon betrieben werden oder welche Eigenschaften die Entitäten haben, spielt für dieses Analysemuster keine Rolle, wodurch dessen generische Anwendbarkeit in unterschiedlichen Szenarien gewährleistet wird. Das Konzept ermöglicht dadurch eine präzise Spezifikation des Kontextes und des Problems, welches durch das automatisierte Analysemuster gelöst werden kann.

7.4 Diskussion und Abgrenzung zu verwandten Arbeiten

Die manuelle Modellierung von Anwendungstopologien und die Konfiguration involvierter Komponenten kann aufgrund von komplexen Abhängigkeiten zu fehlerhaften Modellen führen. Insbesondere wenn abstrakte Geschäftsanforderungen (engl. „*high-level requirements*“) zu konkreten, technischen Konfigurationen und Randbedingungen (engl. „*low-level constraints*“) verfeinert werden müssen, erfordert dies häufig enorme Expertise über unterschiedliche Domänen und Komponententypen. Zur Unterstützung der Modellierung von technisch korrekten Topologiemodellen zur Anwendungsbereitstellung stellen Luo et al. [LLQ07; LLQC08] einen deklarativen, musterbasierten Ansatz vor, um komplexe Konfigurationen und technische Anforderungen (engl. „*constraints*“) automatisiert in Topologiemodellen zu spezifizieren, zu verifizieren, zu analysieren und gegebenenfalls zu korrigieren, falls definierte Randbedingungen verletzt werden. Die automatisierte Spezifikation neuer Konfigurationen bzw. die Korrektur fehlerhafter Modelle wird ähnlich zu dieser Arbeit als Transformation realisiert, jedoch ausschließlich auf Basis von Struktur- und Konfigurationsänderungen. Die Anwendbarkeit eines Musters bzw. die Erkennung von Konfigurationsproblemen wird ausgehend von einer definierten Startkomponente in Form von Navigationspfaden über Komponenten und Relationen sowie durch zu evaluierende Bedingungen bezüglich dieser spezifiziert. Die in dieser Arbeit vorgestellten Analysemuster stellen konzeptionell eine Weiterentwicklung des Ansatzes von Luo dar: Von der strukturellen Modellierung der Anwendungsbereitstellung und deren Konfiguration zur verhaltensorientierten

Modellierung des Anwendungsmanagements. Automatisierte Analysemuster unterscheiden sich somit grundlegend durch (i) unterschiedliche Metamodelle, welche von *Strukturmodellen* (engl. „*structural models*“) zu *Verhaltensmodellen* (engl. „*behavioural models*“) erweitert werden, sowie (ii) in der damit verbundenen unterschiedlichen Spezifikation der Anwendbarkeit und (iii) der Einführung eines expliziten Analysealgorithmus zur Erkennung von Managementfehlern.

Arnold et al. [AEK+07] stellen eine modellbasierte SOA-Deployment-Plattform vor, in der ein Validierungsdienst die korrekte Modellierung von topologiebasierten Zielzustandsmodellen überprüft. Dazu können *Validierungsregeln* spezifiziert werden, die mit einer Domäne von Komponententypen assoziiert sind. Wird die Instantiierung oder Änderung einer Komponente deklariert, werden die zugehörigen Regeln angewendet, welche im Falle eines erkannten Fehlers die jeweilige Stelle im Modell markieren. Ein *Resolution Service* und zugehörige Regeln können dann zur Fehlerbehandlung und Modellkorrektur an den markierten Stellen ausgeführt werden. Während durch dieses Konzept nur zustandsmodifizierende Aufgaben im Rahmen der Bereitstellung und Konfiguration validiert werden können, führt das in dieser Arbeit eingeführte Konzept der automatisierten Analysemuster diesen Ansatz fort und erweitert ihn hinsichtlich der Überprüfung beliebiger Managementaufgaben. Dadurch können auch zustandserhaltende Aufgaben analysiert werden, z. B. ob das Ziel eines Datenexports korrekt angegeben wurde. Zudem wird die Trennung zwischen Fehlererkennung und Fehlerbehebung aufgelöst, wodurch komplexe Fehler durch gezielte, fehlerspezifische Modelltransformationen behandelt werden können. Die Anwendbarkeit von automatisierten Analysemustern wird zudem deutlich feingranularer spezifiziert, indem die zu überprüfende Aufgabe in ihrem Kontext durch das annotierte Topologiefragment modelliert wird.

Scheibenberger und Pansa [SP08] stellen ein Metamodell vor, um Abhängigkeiten (engl. „*dependencies*“) zwischen Anwendungskomponenten und deren Eigenschaften bzw. Zuständen zu beschreiben. In zukünftigen Arbeiten wird untersucht, in wie weit die formale Erfassung der Abhängigkeiten zwischen Eigenschaften von Komponenten und Relationen in DMMN-Modellen die kontextsensitive Analyse unterstützen kann. Eine diesbezügliche Erweiterung des DMMN-Metamodells ist jedoch nicht Gegenstand der vorliegenden Arbeit.

KAPITEL 8

ARCHITEKTUR UND PROTOTYP DES PALMA-FRAMEWORKS

In diesem Kapitel wird die Architektur des PALMA-Frameworks sowie deren prototypische Implementierung beschrieben. Die Architektur umfasst alle Komponenten zur Realisierung der Forschungsbeiträge, welche in der vorliegenden Arbeit vorgestellt wurden, und ermöglicht dadurch die IT-gestützte Ausführung und Automatisierung der PALMA-Methode (siehe [Kapitel 3](#)).

Das Kapitel ist wie folgt aufgebaut: In [Abschnitt 8.1](#) wird die Gesamtarchitektur des PALMA-Frameworks vorgestellt und das Zusammenspiel der individuellen Komponenten erläutert. Insbesondere wird die Integration mit existierenden Management- und Anwendungssystemen diskutiert. In [Abschnitt 8.2](#) wird die prototypische Implementierung der Framework-Architektur vorgestellt und die technische Umsetzung der einzelnen Komponenten detailliert beschrieben. Zudem wird die Wiederverwendbarkeit einzelner Komponenten in anderen Arbeiten aufgezeigt. Die Architektur und prototypische Implementierung stellen den abschließenden [Forschungsbeitrag 6](#) der vorliegenden Arbeit dar.

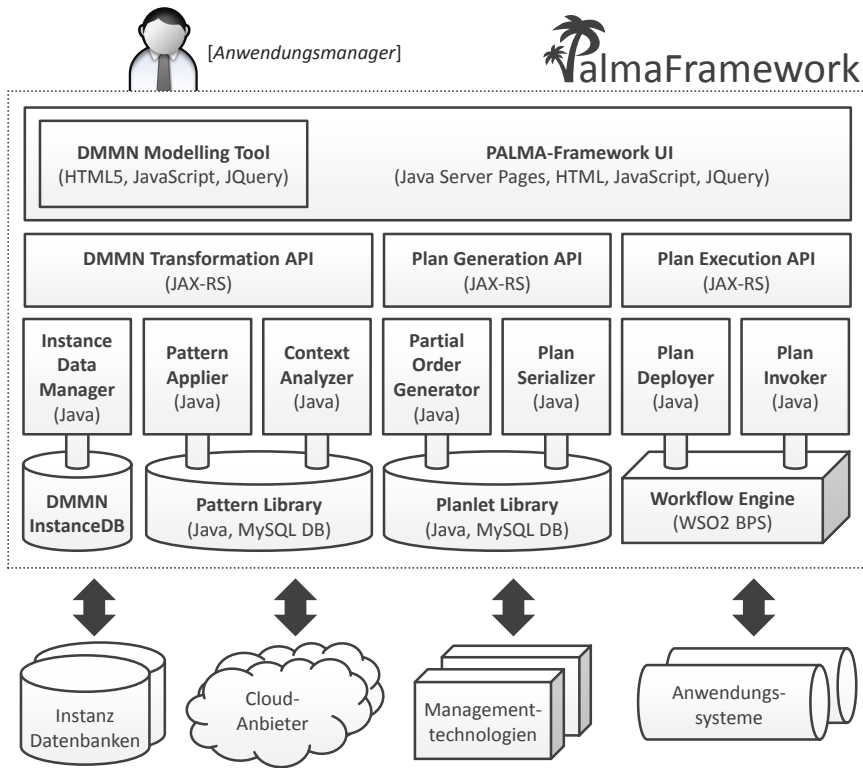


Abbildung 8.1: Architektur des PALMA-Frameworks und technologische Realisierung der prototypischen Implementierung

8.1 Architektur des PALMA-Frameworks

Abbildung 8.1 zeigt die Gesamtarchitektur des PALMA-Frameworks¹. Das Framework kann funktional in zwei Bereiche unterteilt werden, die im Folgenden im Detail erläutert werden: (i) Erstellung von DMMN-Modellen ([Abschnitt 8.1.1](#)) und (ii) Ausführung von DMMN-Modellen ([Abschnitt 8.1.2](#)).

¹Zur besseren Nachvollziehbarkeit werden die englischen Namen der Komponenten verwendet, welche denen der jeweiligen prototypischen Implementierung entsprechen.

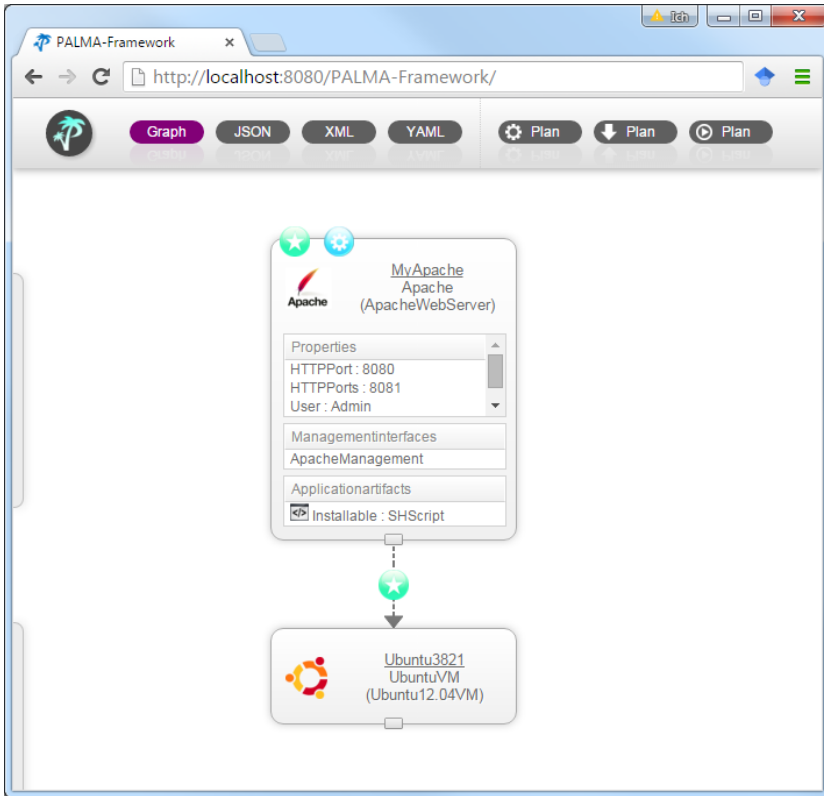


Abbildung 8.2: Grafisches Werkzeug zur Modellierung von DMMN-Modellen

8.1.1 Modellierung, Transformation und Verwaltung von DMMN-Modellen

Das *DMMN-Modelling Tool* und die *PALMA-Framework UI* sind grafische Benutzungsschnittstellen, die es Anwendungsmanagern ermöglichen, (i) DMMN-Modelle zu erstellen, (ii) diese durch die automatisierte Musteranwendung zu transformieren und (iii) zu verwalten. Die grafische Oberfläche des Modellierungswerkzeugs ist in [Abbildung 8.2](#) abgebildet und unterstützt neben der visuellen Modellierung von DMMN-Modellen auch die automatisierte Anwendung von Managementmustern und Analysemustern zur kontextsensitiven

Analyse der deklarierten Managementaufgaben. Dazu nutzt das Werkzeug das *DMMN Transformation API*, um Managementmuster durch die Komponente *Pattern Applier* auf DMMN-Modelle anzuwenden, wodurch die Algorithmen und Konzepte aus [Kapitel 6](#) umgesetzt werden. Gleichmaßen ist das Modellierungswerkzeug über diese Programmierschnittstelle mit dem *Context Analyzer* verbunden, um automatisierte Analysemuster zur Erkennung und Lösung von Managementproblemen, wie in [Kapitel 7](#) konzipiert, einzusetzen. Die beiden Komponenten sind mit einer Bibliothek verbunden, der *Pattern Library*, in der beide Formen automatisierter Muster verwaltet werden. Um diese musterbasierten Funktionalitäten wiederverwendbar zur Verfügung zu stellen, ist das *DMMN Transformation API* gemäß des REST-Architekturstils realisiert und kann in Kombination mit den anderen genannten Komponenten auch außerhalb des PALMA-Frameworks zur Verarbeitung von DMMN-Modellen eingesetzt werden.

Eine weitere Funktionalität der *DMMN Transformation API* ist das Importieren von Instanzmodellen laufender Anwendungen über den *Instance Data Manager*. Dieser Manager stellt einen erweiterbaren Mechanismus zur Verfügung, um Instanzmodelle unterschiedlichen Formats über Plugins zu importieren und in DMMN-Modelle zu transformieren (nicht in [Abbildung 8.1](#) dargestellt). Dadurch können Instanzdatenbanken verschiedener Systeme, deren Schemata auf das topologiebasierte DMMN-Metamodell abbildbar sind, an das PALMA-Framework angeschlossen und Managementaufgaben für die darin repräsentierten Anwendungen ausgeführt werden. Durch diese Normalisierung der Modelle (*Normalizer Pattern* [[HW04](#)]) können die weiteren Komponenten der Architektur agnostisch zu heterogenen Formaten auf Basis von DMMN realisiert werden.

Da DMMN auch zur Modellierung der Anwendungsbereitstellung eingesetzt werden kann, beinhaltet das Framework eine eigene Instanzdatenbank namens *DMMN InstanceDB*. Diese speichert Instanzinformationen über vom Framework selbst bereitgestellte Anwendungen in Form von DMMN-Modellen. Dadurch ermöglicht die Verwendung des DMMN-Formats die Automatisierung (i) der Bereitstellung, (ii) des musterbasierten Managements sowie (iii) der Speicherung von Instanzinformationen betroffener Anwendungen. Die PALMA-Framework UI stellt eine Benutzungsschnittstelle dar, welche diese Funktionalitäten dem Anwendungsmanager anbietet und die weiteren Komponenten integriert.

8.1.2 Ausführung von DMMN-Modellen

Zur Ausführung der deklarierten Managementaufgaben muss das DMMN-Modell, welches durch das Modellierungswerkzeug erstellt wurde, in ein imperatives Managementmodell transformiert und dieses ausgeführt werden. Das REST-basierte *Plan Generation API* stellt dazu Operationen zur Verfügung, welche durch Kombination zweier Komponenten implementiert werden: Der *Partial Order Generator* berechnet eine partielle Ordnung an Management-Planlets zur Ausführung der im DMMN-Modell deklarierten Managementannotationen, wohingegen die resultierende Ordnung anschließend durch den erweiterbaren *Plan Serializer* in eine imperative Workflow-Sprache, beispielsweise BPEL oder BPMN, übersetzt wird. Der *Partial Order Generator* implementiert somit die in [Abschnitt 5.4.7](#) vorgestellten Algorithmen, der *Plan Serializer* beispielsweise die in [Abschnitt 5.5.2](#) vorgestellte Transformation in die Workflow-Sprache BPEL. Durch diese Trennung der Verantwortlichkeiten können unterschiedliche Workflow-Sprachen durch Erweiterung des Serialisierers und der zugehörigen Management-Planlets generisch unterstützt werden. Alle verfügbaren Management-Planlets werden in einer Bibliothek verwaltet, der *Planlet Library*.

Zur Ausführung des generierten Managementplans muss dieser auf einer Ausführungsumgebung ausgeliefert und ausgeführt werden. Dazu stellt das REST-basierte *Plan Execution API* Operationen zur Verfügung, welche mittels zweier weiterer Komponenten realisiert werden: (i) Der *Plan Deployer* installiert das Prozessmodell auf einer Ausführungsumgebung, die das Modell vollautomatisiert ausführen kann. Um die Ausführung des Prozessmodells auszulösen, stellt der (ii) *Plan Invoker* die erforderlichen Operationen bereit. *Plan Deployer* und *Plan Invoker* implementieren dazu, gleichermaßen wie der *Plan Serializer*, einen Erweiterungsmechanismus, der es erlaubt, unterschiedliche Sprachen zur Realisierung imperativer Prozessmodelle per Plugin einzubinden.

Die Architektur des PALMA-Frameworks integriert somit alle Forschungsbeiträge der vorliegenden Arbeit. Zudem können die Framework-Komponenten durch die jeweiligen APIs individuell von anderen Systemen verwendet werden.

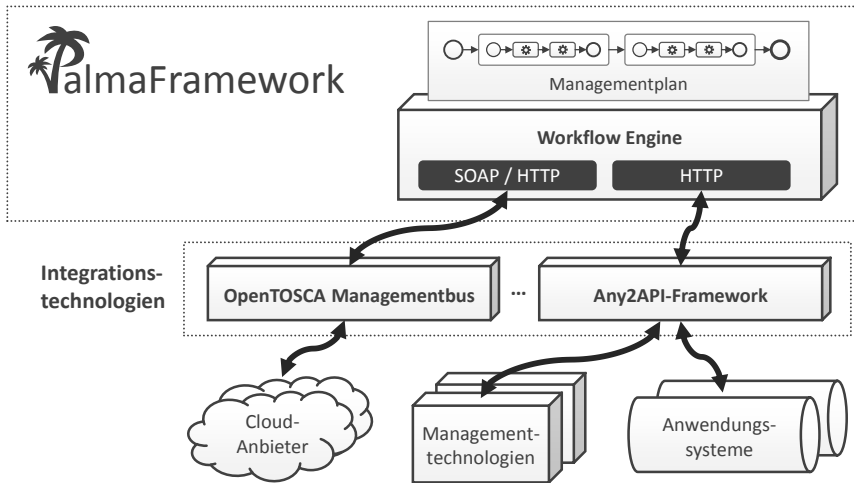


Abbildung 8.3: PALMA-Framework kombiniert mit Integrationsstechnologien

8.1.3 Integration externer Managementsysteme und -komponenten

Das PALMA-Framework implementiert selbst keine technische Managementfunktionalität und realisiert diese ausschließlich über Management-Planlets, welche existierende Managementtechnologien, Werkzeuge und Schnittstellen von Anwendungssystemen orchestrieren. Um die Komplexität der Planlet-Entwicklung zu reduzieren, kann das Framework mit *Integrationstechnologien* kombiniert werden, welche speziell für das Anwendungsmanagement entwickelt wurden. Das Cloud-Managementsystem *OpenTOSCA* [BBH+13] beinhaltet beispielsweise einen Managementbus [WBB+14b; WBB+14a], während das *Any2API-Framework* [WBL15a] existierende Technologien automatisiert als APIs kapselt. Mittels dieser Integrationstechnologien können aufrufbare Dienste bereitgestellt werden, welche proprietäre Technologien auf Standards wie TOSCA und HTTP abbilden. Zudem können durch diese Ansätze die in Abschnitt 2.3.2 analysierten Probleme bezüglich sich ändernder APIs abgeschwächt werden, da diese Integrationstechnologien eine weitere Indirektion einführen und derart Integrationsprobleme somit lokaler behandelt werden können.

8.1.4 Abgrenzung zu verwandten Arbeiten

Die von Maghraoui et al. [EME+06] vorgestellte Architektur eines Bereitstellungssystems, welches aus deklarativen Zielzustandsmodellen imperative Workflow-Modelle generiert und diese auf einer Laufzeitumgebung ausführt, wird in der vorliegenden Arbeit um Komponenten zur Automatisierung von Management- und Analysemustern erweitert. Zudem wird die Architektur hinsichtlich der Datenflüsse adaptiert, um das Management von Anwendungen durch den Zugriff auf die jeweiligen Instanzmodelle zu ermöglichen.

8.2 Prototypische Implementierung des PALMA-Frameworks

In diesem Abschnitt werden die technischen Details der prototypischen Implementierung des PALMA-Frameworks vorgestellt, die auf der in [Abbildung 8.1](#) vorgestellten Zwei-Schichten-Architektur basiert.

Die grafische Benutzungsoberfläche, d. h. das DMMN Modelling Tool und die PALMA-Framework UI, sind basierend auf Web 2.0 Technologien realisiert (HTML 5, CSS 3, JavaScript, JQuery¹) und Server-seitig als Java Server Pages (JSP)² implementiert. Die Benutzungsoberfläche wird daher in einem Browser angezeigt, wodurch keine zusätzlichen Softwarekomponenten seitens des Anwendungsmanagers installiert werden müssen. Die Oberfläche kommuniziert ausschließlich über HTTP mit den REST-basierten PALMA-Framework-APIs, wodurch ein Großteil der Geschäftslogik auf dem Server betrieben wird.

Die APIs sind als Java-basierte Web Services realisiert, die mittels JAX-RS³ implementiert wurden. Alle Web Services werden auf einem Tomcat 7-Webserver⁴ betrieben. Die Bibliotheken sind als Web Services in Kombination mit MySQL-Datenbanken realisiert, die Meta-Informationen über die Artefakte verwalten. Die zugehörigen Dateien, d. h. Prozessmodelle und Mustertransformationen, werden über das Dateisystem des Servers verwaltet.

¹<https://jquery.com/>

²<http://www.oracle.com/technetwork/java/javaee/jsp/>

³http://download.oracle.com/otndocs/jcp/jaxrs-2_0-fr-eval-spec/

⁴<http://tomcat.apache.org/download-70.cgi>

8.2.1 Serialisierung von DMMN-Modellen

DMMN-Modelle werden vorwiegend auf Basis der *Extensible Markup Language (XML)* serialisiert, welche sich aufgrund der Standardisierung, der Möglichkeit explizite Metamodelle zu definieren und der Verfügbarkeit zahlreicher Bibliotheken in Java besonders dazu eignen, maschinenlesbare Modelle zu serialisieren und diese effizient zu verarbeiten. Dazu wurde das DMMN-Metamodell mittels Java-Klassen modelliert und die Serialisierung durch Annotationen definiert, wodurch DMMN-Modelle vollautomatisch mittels JAXB¹ eingelesen und exportiert werden können. Topologiefragmente von Planlets und automatisierten Mustern werden daher in dieser Arbeit ebenfalls in XML modelliert. Zusätzlich unterstützt der Prototyp DMMN-Repräsentationen in der *JavaScript Object Notation (JSON)* und der *YAML Ain't Markup Language (YAML)*.

8.2.2 Implementierung und Ausführung von Planlets und Plänen

Die imperativen Managementmodelle von Management-Planlets sowie die generierten Managementpläne werden gemäß [Abschnitt 5.5.2](#) in BPEL realisiert und folgen den definierten Konzepten zur Fehlerbehandlung und Kompensation. Zur Ausführung der generierten Managementpläne wird die BPEL-Laufzeitumgebung *WSO2 Business Process Server (BPS)*² eingesetzt, welche unter anderem das Konzept der Extension Activities [[OAS07b](#)] unterstützt und somit die Integration externer Management- und Integrationstechnologien ermöglicht, etwa um REST-basierte Dienstaufrufe, wie von Haupt et al. [[HFK+14](#)] vorgestellt, zu unterstützen. Dies fordert vor allem die Interaktion mit Cloud-Anbietern, die typischerweise (REST-basierte) APIs anbieten, die über HTTP aufgerufen werden können. Durch die native Unterstützung von SOAP / HTTP des BPS kann das Cloud-Managementsystem OpenTOSCA [[BBH+13](#)] ebenfalls wie in [Abschnitt 8.1.3](#) beschrieben in den Prototyp zur Ausführung von Managementoperationen eingebunden werden. Dazu werden spezielle CSARs³

¹Java Architecture for XML Binding

²<http://wso2.com/products/business-process-server/>

³Cloud Service Archive [[OAS13b](#)]

eingesetzt, welche TOSCA Node Types enthalten, die von Planlets benötigte Funktionen über den OpenTOSCA-internen Managementbus zugänglich machen. Die OpenTOSCA Laufzeitumgebung [BBH+13] wird daher zum Betrieb dieser CSARs eingesetzt, während das TOSCA-Modellierungswerkzeug *Winery* [KBBL13] deren Modellierung und Paketierung ermöglicht.

8.2.3 Realisierung und Implementierung automatisierter Muster

Die Umsetzung der PALMA-Methode soll unabhängig von individuellen Technologien sowie deren Formaten sein und diesbezüglich flexibel erweitert werden können, was durch [Anforderung 2](#) in [Abschnitt 3.2](#) ausdrücklich gefordert wird. Da Anwendungsinstanzmodelle in Abhängigkeit der eingesetzten Technologien, welche diese erstellen, aktualisieren und verwalten, auf unterschiedlichen Formaten und Metamodellen basieren, werden automatisierte Muster ausschließlich auf Basis eines normalisierten Metamodells in Form von DMMN implementiert. Dies verhindert die Kopplung an unterschiedliche, proprietäre Metamodelle für Anwendungsinstanzen und ermöglicht die Unterstützung verschiedener Technologien durch ein uniformes Format, das aufgrund seiner Generizität und Erweiterbarkeit hinsichtlich Komponenten- und Relationentypen eine systematische Normalisierung ermöglicht. Daher werden automatisierte Muster ausschließlich auf DMMN-Modelle angewendet, auch wenn diese nur den aktuellen Zustand der jeweiligen Anwendungsinstanz repräsentieren. Dies hat den weiteren Vorteil, dass Mustertransformationen nur für eine Sprache implementiert werden müssen. Instanzmodelle in anderen Formaten werden daher während des Imports durch das DMMN Management API in die Topologie- und technische Managementschicht von DMMN übersetzt und anschließend dem automatisierten Managementmuster übergeben. Die Implementierung der Mustertransformation sowie der Analysealgorithmen zur Erkennung von Managementfehlern wird in der prototypischen Implementierung mittels Java umgesetzt. Die Implementierungen werden über einen Reflection-basierten Erweiterungsmechanismus integriert. Die Topologiefragmente automatisierter Muster werden gemäß [Abschnitt 8.2.1](#) in XML beschrieben.

8.3 Verwendung des PALMA-Frameworks in anderen Arbeiten

In diesem Abschnitt werden zwei Arbeiten vorgestellt, welche Management-Planlets ([Forschungsbeitrag 3](#)) sowie automatisierte Managementmuster ([Forschungsbeitrag 4](#)) zur Unterstützung der Musteranwendung einsetzen.

8.3.1 Automatisierung von Green Business Process Patterns

Nowak et al. [[NBL14](#)] stellen einen Ansatz vor, um die CO₂ Emission einer virtuellen Maschine durch automatisierte Anwendung des *Green Compensation Patterns* [[NLS+11](#)] zu kompensieren. Das Green Compensation Pattern wurde dazu als automatisiertes Managementmuster modelliert und kann auf Anwendungen angewendet werden, welche virtuelle Maschinen mit einem Linux-basierten Betriebssystem einsetzen. Das Muster erzeugt ein DMMN-Modell, das die Installation einer Überwachungskomponente auf der Maschine deklariert, welche bei Überschreiten eines bestimmten Schwellenwertes hinsichtlich der CPU-Auslastung einen Kompensationsdienst aufruft. Die Anwendung des in dieser Arbeit vorgestellten Musterkonzepts in diesem Szenario zeigt, dass das Konzept auch zum Management und zur Optimierung laufender Anwendungen hinsichtlich ökologischer Faktoren eingesetzt werden kann.

8.3.2 Wiederverwendung bewährter Lösungsimplementierungen

Die Anwendung eines abstrakten Managementmusters auf einen konkreten Anwendungsfall erfordert die Verfeinerung der beschriebenen Lösung (siehe [Abschnitt 2.4.2.1](#)). Falkenthal et al. [[FBB+14b](#); [FBB+14a](#)] stellen Ansätze vor, die sogenannte *Solution Implementations* als wiederverwendbare, direkt ausführbare Lösungsimplementierungen eines bestimmten Musters einsetzen. Das Konzept der Management-Planlets wurde in diesen Arbeiten dazu verwendet, manuell ausführbare Lösungsimplementierungen in Form von Managementplänen zu erstellen, welche aus individuellen Planlets zusammengesetzt sind, und diese mit den jeweiligen Managementmustern zu assoziieren. Die automatisierte Plangenerierung durch die vorliegende Arbeit erweitert dieses Konzept.

ZUSAMMENFASSUNG UND AUSBLICK

In der vorliegenden Dissertation wurde mit der PALMA-Methode ein Vorgehensmodell eingeführt, das die systematische Automatisierung und Integration bewährten Managementwissens unterschiedlicher Abstraktionshöhen ermöglicht. Während Management-Planlets *technisches Managementwissen* in Form modularer, ausführbarer Prozessmodelle beschreiben, ermöglichen automatisierte Managementmuster die Automatisierung bewährten *konzeptionellen Managementwissens* durch Modelltransformationen. Die eingeführte deklarative Modellierungssprache DMMN, welche zur technologieunabhängigen Modellierung beliebiger Managementziele eingesetzt werden kann, dient dabei als Grundlage, um Musterlösungen in Form von DMMN-Modellen automatisiert für individuelle Anwendungen zu generieren und diese durch Orchestrierung geeigneter Planlets auszuführen. Durch die technologieunabhängige Zielbeschreibung mittels DMMN wird die Komplexität des Anwendungsmanagements signifikant reduziert und die Automatisierung abstrakter Muster ermöglicht.

Die vollständige Automatisierbarkeit der PALMA-Methode durch existierende Arbeiten sowie durch die weiteren Forschungsbeiträge der vorliegenden Dissertation reduziert systematisch manuelle Fehlerquellen des Anwendungsmanagements, da sowohl die Modellierung zu erreichender Managementziele als auch deren Ausführung durch das PALMA-Framework unterstützt und automatisiert werden. Die Reduktion manueller Managementfehler wird zudem durch die deklarative Modellierungssprache DMMN unterstützt, die neben einem formalen topologiebasierten Metamodell zusätzlich ein grafisches Metamodell definiert, welches die Grundlage für eine intuitive DMMN-Modellierung legt. Des Weiteren wurde mit den automatisierten Analysemustern ein Konzept vorgestellt, mithilfe dessen Managementfehler durch Automatisierung bewährten Wissens präventiv erkannt und behoben werden können. Durch die explizite, von der PALMA-Methode definierte, Transformation deklarativer Managementmodelle in imperative Prozessmodelle und dem damit ermöglichten Einsatz der Workflow-Technologie wird zudem eine verlässliche und robuste Ausführung der modellierten Managementziele gewährleistet. Dies wurde im Rahmen der vorliegenden Dissertation auf Basis der Workflow-Sprache BPEL validiert.

Durch Kombination der vorgestellten Konzepte unterstützt die PALMA-Methode die Kooperation verschiedener Experten, die sich durch eine klare Trennung der Verantwortlichkeiten auf ihre jeweilige Kernkompetenz fokussieren können, deren Ergebnisse jedoch automatisiert integriert werden. Durch das entstandene PALMA-Framework können Organisationen somit konzeptionelle und technische Managementexpertise unabhängig von individuellen Anwendungen erfassen und homogen formalisieren, nicht vorhandenes Wissen beider Ebenen zukaufen sowie dieses automatisiert für individuelle Anwendungsfälle ausführen.

Zukünftige Forschungsarbeiten im Rahmen der PALMA-Methode und der weiteren Beiträge dieser Dissertation können die Effizienz des Anwendungsmanagements zusätzlich steigern. Diese werden im Folgenden skizziert sowie mögliche Vorteile gegenüber der Realisierung in dieser Arbeit diskutiert.

Die Entwicklung von Management-Planlets erfordert typischerweise die Integration unterschiedlicher heterogener Managementtechnologien. Beispielsweise, um APIs von Cloud-Anbietern oder Konfigurationsmanagementwerkzeuge zu orchestrieren. Obwohl diese Integration nur einmalig bei der Entwicklung eines Planlets durchgeführt werden muss, beinhaltet dies ein wesentliches Optimierungspotenzial. Während beispielsweise die Ansätze von Wettinger et al. [WBB+14b; WBB+14a; WBL15a] die Integration einzelner Technologien signifikant vereinfachen (vgl. [Abschnitt 8.1.3](#)), ist das Fehlen optimierter Modellierungskonstrukte in Workflow-Sprachen zum Aufruf unterschiedlicher Managementtechnologien ein fundamentales Problem, das weitestgehend individuell gelöst werden muss. Ein erster Ansatz, um deklarative Konfigurationsmanagementwerkzeuge nahtlos in imperative Workflow-Modelle zu integrieren, wurde von Breitenbücher et al. [BBK+15a] vorgestellt. Dieser Ansatz kann in zukünftigen Arbeiten mit DMMN kombiniert werden, um deklarative Managementaktivitäten innerhalb eines imperativen Prozessmodells zu spezifizieren.

Das Management von Multi-Cloud-Anwendungen, die auf mehrere Cloud-Umgebungen verteilt sind, führt durch die zentrale Ausführung des generierten Managementplans zu erhöhtem Kommunikationsaufwand zwischen Plan und den jeweiligen Umgebungen. In zukünftigen Forschungsarbeiten kann daher untersucht werden, wie Managementchoreographien aus DMMN-Modellen generiert werden können, die lokal in den jeweiligen Cloud-Umgebungen ausgeführt werden und nur in notwendigen Schritten miteinander kommunizieren.

DMMN ermöglicht die deklarative Modellierung beliebiger Managementaufgaben, indem diese durch Annotationen direkt an den betroffenen Komponenten- und Relationenelementen des Modells spezifiziert werden. Dies erfordert, in Abhängigkeit des Managementumfangs, eine Vielzahl einzelner Modellierungsschritte, falls keine automatisierten Managementmuster eingesetzt werden. In zukünftigen Forschungsarbeiten kann dieser Aufwand reduziert werden, indem DMMN um Konstrukte erweitert wird, welche die deklarative Spezifikation höherwertiger Aufgaben und Einschränkungen ermöglichen. Beispielsweise, um die Migration einer gesamten laufenden LAMP-Infrastruktur in eine andere Umgebung durch ein einzelnes deklaratives Managementziel zu modellieren.

DANKSAGUNGEN

Mein besonderer Dank gilt an erster Stelle meinem Doktorvater Prof. Dr. Dr. h. c. Frank Leymann für die Betreuung und Unterstützung meiner Promotion. Vielen Dank für die Möglichkeit am IAAS arbeiten und forschen zu dürfen, die zahlreichen und wertvollen Diskussionen, sowie für das entgegengebrachte Vertrauen, den überlassenen Freiraum und die übertragene Verantwortung. Des Weiteren möchte ich mich sehr herzlich bei meinem Zweitgutachter Univ. Prof. Dr. Schahram Dustdar und meinem Drittgutachter Prof. Dr. Florian Daniel für die Übernahme des Mitberichts bedanken.

Ein großes Dankeschön möchte ich den Kollegen aussprechen, die mich im Rahmen meiner wissenschaftlichen Arbeit begleitet haben. Insbesondere danke ich meinen Kollegen und Freunden Oliver Kopp, Tobias Binz und Johannes Wettinger für die zahlreichen Diskussionen und die vielfältige Unterstützung.

Mein ganz besonderer Dank gilt meiner Familie. Vielen Dank meinen Eltern Elke und Fritz, die mich zu jedem Zeitpunkt meines Lebens unterstützt und mir diesen Weg ermöglicht haben. Vielen Dank meiner Schwester Jasmin, die mir in zahlreichen Situationen den Rücken freigehalten hat. Vielen Dank meiner Frau Sandra für ihre nicht nachlassende Geduld und die großartige Unterstützung.

LITERATURVERZEICHNIS

- [AEK+07] W. Arnold, T. Eilam, M. Kalantar, A. V. Konstantinou und A. A. Totok. „Pattern Based SOA Deployment“. In: *Proceedings of the Fifth International Conference on Service-Oriented Computing (ICSOC 2007)*. Springer, Sep. 2007, S. 1–12 (Zitiert auf S. [82](#), [267](#), [285](#)).
- [AEK+08] W. Arnold, T. Eilam, M. Kalantar, A. V. Konstantinou und A. A. Totok. „Automatic Realization of SOA Deployment Patterns in Distributed Environments“. In: *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC 2008)*. Springer, Dez. 2008, S. 162–179 (Zitiert auf S. [267](#)).
- [AGLW14] V. Andrikopoulos, S. Gómez Sáez, F. Leymann und J. Wettinger. „Optimal Distribution of Applications in the Cloud“. In: *Proceedings of the 26th International Conference on Advanced Information Systems Engineering (CAiSE 2014)*. Springer, Juni 2014, S. 75–90 (Zitiert auf S. [27](#)).
- [AHS05] A. Andrzejak, U. Hermann und A. Sahai. „FEEDBACKFLOW - An Adaptive Workflow Generator for Systems Management“. In: *Proceedings of the Second International Conference on Autonomic Computing (ICAC 2005)*. Juni 2005, S. 335–336 (Zitiert auf S. [57](#)).
- [AHW03] N. Arshad, D. Heimbigner und A. Wolf. „Deployment and Dynamic Reconfiguration Planning For Distributed Software Systems“. In: *Proceedings of the 15th IEEE International Conference on Tools with Artificial*

- Intelligence (ICTAI 2003)*. IEEE, Nov. 2003, S. 39–46 (Zitiert auf S. 57, 87).
- [AHW07] N. Arshad, D. Heimbigner und A. L. Wolf. „Deployment and dynamic reconfiguration planning for distributed software systems“. In: *Software Quality Journal* 15.3 (2007), S. 265–281 (Zitiert auf S. 57, 87).
- [AIS77] C. Alexander, S. Ishikawa und M. Silverstein. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, Aug. 1977 (Zitiert auf S. 90).
- [Ale79] C. Alexander. *The Timeless Way of Building*. Oxford University Press, 1979 (Zitiert auf S. 90).
- [Ama16] Amazon. *AWS CloudFormation*. Feb. 2016. URL: <https://aws.amazon.com/de/cloudformation/> (Zitiert auf S. 66).
- [APM16] APM Group Ltd. *Official ITIL Website*. Feb. 2016. URL: <http://www.itil-officialsite.com> (Zitiert auf S. 40).
- [AtHEvA06] M. Adams, A. H. M. ter Hofstede, D. Edmond und W. M. P. van der Aalst. „Worklets: A Service-oriented Implementation of Dynamic Flexibility in Workflows“. In: *On the Move to Meaningful Internet Systems: OTM 2006 (CoopIS 2006)*. Springer, Nov. 2006, S. 291–308 (Zitiert auf S. 71).
- [AZ05] P. Avgeriou und U. Zdun. „Architectural Patterns Revisited – A Pattern Language“. In: *In 10th European Conference on Pattern Languages of Programs (EuroPlop 2005)*. UVK - Universitaetsverlag Konstanz, Juli 2005 (Zitiert auf S. 91).
- [Bae73] J. L. Baer. „A Survey of Some Theoretical Aspects of Multiprocessing“. In: *ACM Computing Surveys* 5.1 (März 1973), S. 31–80 (Zitiert auf S. 216).
- [Bai83] L. Bainbridge. „Ironies of Automation“. In: *Automatica*. Bd. 19. 6. Pergamon Press Ltd., Mai 1983, S. 775–779 (Zitiert auf S. 46).
- [BBF+14] U. Breitenbücher, T. Binz, C. Fehling, O. Kopp, F. Leymann und M. Wieland. „Policy-Aware Provisioning and Management of Cloud Applications“. In: *International Journal On Advances in Security* 7.1&2 (2014) (Zitiert auf S. 24, 164).

- [BBH+13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak und S. Wagner. „OpenTOSCA - A Runtime for TOSCA-based Cloud Applications“. In: *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*. Springer, Dez. 2013, S. 692–695 (Zitiert auf S. 70, 73, 292, 294, 295).
- [BBK+12] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und D. Schumm. „Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA“. In: *On the Move to Meaningful Internet Systems: OTM 2012 (CoopIS 2012)*. Springer, Sep. 2012, S. 416–424 (Zitiert auf S. 23, 30, 32, 36, 155, 164).
- [BBK+13a] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und J. Wettinger. „Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies“. In: *On the Move to Meaningful Internet Systems: OTM 2013 Conferences (CoopIS 2013)*. Springer, Sep. 2013, S. 130–148 (Zitiert auf S. 13, 23, 42, 45, 50, 75, 77).
- [BBK+13b] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und M. Wieland. „Policy-Aware Provisioning of Cloud Applications“. In: *Proceedings of the Seventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2013)*. Xpert Publishing Services, Aug. 2013, S. 86–95 (Zitiert auf S. 24, 164).
- [BBK+14a] U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann und J. Wettinger. „Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA“. In: *International Conference on Cloud Engineering (IC2E 2014)*. IEEE, März 2014, S. 87–96 (Zitiert auf S. 23, 27, 32, 45, 70).
- [BBK+14b] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und M. Wieland. „Context-Aware Cloud Application Management“. In: *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014)*. SciTePress, Apr. 2014, S. 499–509 (Zitiert auf S. 23, 28, 37, 42, 75, 82, 83, 108).
- [BBK+15a] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und J. Wettinger. „A Modelling Concept to Integrate Declarative and Imperative Cloud Application Provisioning Technologies“. In: *Proceedings of the 5th International*

Conference on Cloud Computing and Services Science (CLOSER 2015). SciTePress, Mai 2015, S. 487–496 (Zitiert auf S. 24, 79, 231, 299).

- [BBK+15b] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann und M. Wieland. „Context-Aware Provisioning and Management of Cloud Applications“. In: *Cloud Computing and Services Science*. Communications in Computer and Information Science. Springer, 2015, S. 151–168 (Zitiert auf S. 23, 28).
- [BBKL13a] T. Binz, U. Breitenbücher, O. Kopp und F. Leymann. „Automated Discovery and Maintenance of Enterprise Topology Graphs“. In: *Proceedings of the 6th IEEE International Conference on Service Oriented Computing and Applications (SOCA 2013)*. IEEE, Dez. 2013, S. 126–134 (Zitiert auf S. 29, 105, 106).
- [BBKL13b] U. Breitenbücher, T. Binz, O. Kopp und F. Leymann. „Pattern-based Runtime Management of Composite Cloud Applications“. In: *Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013)*. SciTePress, Mai 2013, S. 475–482 (Zitiert auf S. 22, 70, 77).
- [BBKL14a] T. Binz, U. Breitenbücher, O. Kopp und F. Leymann. „Migration of enterprise applications to the cloud“. In: *it - Information Technology, Special Issue: Architecture of Web Application* 56.3 (Mai 2014), S. 106–111 (Zitiert auf S. 28).
- [BBKL14b] T. Binz, U. Breitenbücher, O. Kopp und F. Leymann. „TOSCA: Portable Automated Deployment and Management of Cloud Applications“. In: *Advanced Web Services*. Springer, Jan. 2014, S. 527–549 (Zitiert auf S. 27, 32).
- [BBKL14c] U. Breitenbücher, T. Binz, O. Kopp und F. Leymann. „Automating Cloud Application Management Using Management Idioms“. In: *Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014)*. Xpert Publishing Services, Mai 2014, S. 60–69 (Zitiert auf S. 13, 23, 42, 94, 258).
- [BBKL14d] U. Breitenbücher, T. Binz, O. Kopp und F. Leymann. „Vinothek - A Self-Service Portal for TOSCA“. In: *Proceedings of the 6th Central-European Workshop on Services and their Composition (ZEUS 2014)*. CEUR-WS.org, Feb. 2014, S. 69–72 (Zitiert auf S. 73).

- [BBL14] U. Breitenbücher, T. Binz und F. Leymann. „A Method to Automate Cloud Application Management Patterns“. In: *Proceedings of the Eighth International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2014)*. Xpert Publishing Services, Aug. 2014, S. 140–145 (Zitiert auf S. 23, 266).
- [BBLS12] T. Binz, G. Breiter, F. Leymann und T. Spatzier. „Portable Cloud Services Using TOSCA“. In: *IEEE Internet Computing* 16.03 (Mai 2012), S. 80–85 (Zitiert auf S. 32).
- [BCFP13] P. Bellavista, A. Corradi, L. Foschini und A. Pernaflni. „Towards an Automated BPEL-based SaaS Provisioning Support for OpenStack IaaS“. In: *Scalable Computing* 14.4 (2013), S. 235–247 (Zitiert auf S. 70, 79).
- [BCK03] L. Bass, P. Clements und R. Kazman. *Software Architecture in Practice*. Addison-Wesley, Apr. 2003 (Zitiert auf S. 26).
- [BCP+01] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri und P. Traverso. „MBP: a Model Based Planner“. In: *Proceedings of the IJCAI-01 Workshop on Planning under Uncertainty and Incomplete Information*. AAAI, 2001 (Zitiert auf S. 57).
- [BDtH05] A. Barros, M. Dumas und A. H. M. ter Hofstede. „Service Interaction Patterns“. In: *Proceedings of the 3rd International Conference on Business Process Management (BPM 2005)*. Springer, Sep. 2005, S. 302–318 (Zitiert auf S. 91, 230).
- [Bec07] K. Beck. *Implementation Patterns*. 1. Aufl. Addison-Wesley, Nov. 2007 (Zitiert auf S. 91).
- [Bec96] K. Beck. *SmallTalk Best Practice Patterns*. Prentice Hall, Okt. 1996 (Zitiert auf S. 91).
- [Ber66] A. Bernstein. „Analysis of Programs for Parallel Processing“. In: *Transactions on Electronic Computers* 5 (Okt. 1966), S. 757–763 (Zitiert auf S. 216).
- [BFL+12] T. Binz, C. Fehling, F. Leymann, A. Nowak und D. Schumm. „Formalizing the Cloud through Enterprise Topology Graphs“. In: *Proceedings of 2012 IEEE International Conference on Cloud Computing (CLOUD 2012)*. IEEE, Juni 2012, S. 742–749 (Zitiert auf S. 28, 115, 164).

- [BH05] A. B. Brown und J. L. Hellerstein. „Reducing the cost of IT operations: is automation always the answer?“ In: *Proceedings of the 10th Conference on Hot Topics in Operating Systems (HOTOS 2005)*. USENIX, Juni 2005 (Zitiert auf S. 41, 45, 46).
- [Bie10] M. Biehl. *Literature Study on Model Transformations*. Techn. Ber. Royal Institute of Technology, Juli 2010 (Zitiert auf S. 252, 253).
- [Bin15] T. Binz. „Crawling von Enterprise Topologien zur automatisierten Migration von Anwendungen: eine Cloud-Perspektive“. Dissertation. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2015 (Zitiert auf S. 26, 28, 29, 105, 106, 115, 164).
- [BK06] A. Brown und A. Keller. „A Best Practice Approach for Automating IT Management Processes“. In: *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*. IEEE, Apr. 2006, S. 33–44 (Zitiert auf S. 43).
- [BKH05] A. Brown, A. Keller und J. Hellerstein. „A model of configuration complexity and its application to a change management system“. In: *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)*. IEEE, Mai 2005, S. 631–644 (Zitiert auf S. 42).
- [BMM12] E. Brandtzæg, P. Mohagheghi und S. Mosser. „Towards a domain-specific language to deploy applications in the clouds“. In: *Proceedings of the Third International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2012)*. Xpert Publishing Services, Juli 2012, S. 213–218 (Zitiert auf S. 32).
- [BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad und M. Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, Okt. 1996 (Zitiert auf S. 13, 90, 91, 258).
- [BP01] A. B. Brown und D. A. Patterson. „To Err is Human“. In: *Proceedings of the First Workshop on Evaluating and Architecting System dependability (EASY 2001)*. Juli 2001 (Zitiert auf S. 13, 41).
- [BRJ05] G. Booch, J. Rumbaugh und I. Jacobson. *The Unified Modeling Language User Guide (2nd Edition)*. Addison-Wesley, Mai 2005 (Zitiert auf S. 26).

- [BTN+14] A. Bergmayr, J. Troya, P. Neubauer, M. Wimmer und G. Kappel. „UML-based Cloud Application Modeling with Libraries, Profiles, and Templates“. In: *Proceedings of the 2nd International Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE 2014)*. CEUR-WS.org, Sep. 2014, S. 56–65 (Zitiert auf S. 34).
- [Bur95] M. Burgess. „A Site Configuration Engine“. In: *Computing Systems* 8.2 (1995), S. 309–337 (Zitiert auf S. 52).
- [BWG+10] L. A. Bianchin, J. A. Wickboldt, L. Z. Granville, L. P. Gaspary, C. Bartolini und M. Rahmouni. „Similarity Metric for Risk Assessment in IT Change Plans“. In: *Proceedings of the 6th International Conference on Network and Service Management (CNSM 2010)*. IEEE, Okt. 2010, S. 25–32 (Zitiert auf S. 83).
- [BZ95] R. Boutaba und S. Znaty. „An Architectural Approach for Integrated Network and Systems Management“. In: *SIGCOMM Computer Communication Review* 25.5 (Okt. 1995), S. 13–38 (Zitiert auf S. 68, 118).
- [Can07] D. Cannon. *ITIL - Service Operation*. The Stationery Office, Mai 2007 (Zitiert auf S. 29, 41, 73, 117).
- [CEM+10] C. Chapman, W. Emmerich, F. G. Márquez, S. Clayman und A. Galis. „Software Architecture Definition for On-demand Cloud Provisioning“. In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC 2010)*. ACM, Juni 2010, S. 61–72 (Zitiert auf S. 33).
- [CFSV01] L. P. Cordella, P. Foggia, C. Sansone und M. Vento. „An Improved Algorithm for Matching Large Graphs“. In: *Proceedings of the 3rd International Workshop on Graph-Based Representations in Pattern Recognition (IAPR-TC15)*. Jan. 2001, S. 149–159 (Zitiert auf S. 200).
- [CFSV04] L. P. Cordella, P. Foggia, C. Sansone und M. Vento. „A (sub) graph isomorphism algorithm for matching large graphs“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.10 (2004), S. 1367–1372 (Zitiert auf S. 200, 208).
- [CFSV98] L. P. Cordella, P. Foggia, C. Sansone und M. Vento. „Subgraph Transformations for the Inexact Matching of Attributed Relational Graphs“. In: *Graph Based Representations in Pattern Recognition*. Bd. 12. Computing Supplement. Springer, 1998, S. 43–52 (Zitiert auf S. 200).

- [CFSV99] L. P. Cordella, P. Foggia, C. Sansone und M. Vento. „Performance evaluation of the VF graph matching algorithm“. In: *Proceedings of the 10th International Conference on Image Analysis and Processing (ICIAP 1999)*. IEEE, Sep. 1999, S. 1172–1177 (Zitiert auf S. 200).
- [CGND10] F. Cuadrado, R. García-Carmona, Á. Navas und J. Dueñas. „A Change Execution System for Enterprise Services with Compensation Support“. In: *Proceedings of the International Conference on ENTERprise Information Systems (CENTERIS 2010)*. Springer, Okt. 2010, S. 441–450 (Zitiert auf S. 86).
- [Cha08] D. Chappell. *What is Application Lifecycle Management*. Dez. 2008. URL: <http://www.davidchappell.com/WhatIsALM--Chappell.pdf> (Zitiert auf S. 38, 39).
- [Che76] P. P.-s. Chen. „The Entity-Relationship Model: Toward a Unified View of Data“. In: *ACM Transactions on Database Systems* 1 (März 1976), S. 9–36 (Zitiert auf S. 36).
- [CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest und C. Stein. *Introduction to Algorithms, Second Edition*. MIT Press, Sep. 2001 (Zitiert auf S. 197).
- [Cop96] J. O. Coplien. *Software Patterns*. SIGS Books & Multimedia, 1996 (Zitiert auf S. 258).
- [CZM+11] Y.-M. Chee, N. Zhou, F. J. Meng, S. Bagheri und P. Zhong. „A Pattern-Based Approach to Cloud Transformation“. In: *Proceedings of the 2011 International Conference on Cloud Computing (CLOUD 2011)*. IEEE, Juli 2011, S. 388–395 (Zitiert auf S. 96, 266).
- [DBC+12] F. D’Andria, S. Bocconi, J. G. Cruz, J. Ahtes und D. Zeginis. „Cloud4SOA: Multi-cloud Application Management Across PaaS Offerings“. In: *Proceedings of the 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2012)*. IEEE, Sep. 2012, S. 407–414 (Zitiert auf S. 50).
- [dCMA+08] W. L. da Costa Cordeiro, G. S. Machado, F. G. Andreis, A. D. Santos, C. B. Both, L. P. Gaspary, L. Z. Granville, C. Bartolini und D. Trastour. „A Runtime Constraint-Aware Solution for Automated Refinement of IT Change Plans“. In: *Managing Large-Scale Service Deployment*. Bd. 5273. Springer, 2008, S. 69–82 (Zitiert auf S. 40, 73).

- [dCMA+09] W. L. da Costa Cordeiro, G. S. Machado, F. G. Andreis, A. D. dos Santos, C. B. Both, L. P. Gasparly, L. Z. Granville, C. Bartolini und D. Trastour. „ChangeLedge: Change design and planning in networked systems based on reuse of knowledge and automation“. In: *Computer Networks* 53.16 (2009), S. 2782–2799 (Zitiert auf S. 40, 47, 73).
- [dCMD+08] W. L. da Costa Cordeiro, G. S. Machado, F. F. Daitx, C. B. Both, L. P. Gasparly, L. Z. Granville, A. Sahai, C. Bartolini, D. Trastour und K. Srikoski. „A Template-based Solution to Support Knowledge Reuse in IT Change Design“. In: *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2008)*. IEEE, Apr. 2008, S. 355–362 (Zitiert auf S. 40, 73).
- [DJ07] T. Delaet und W. Joosen. „PoDIM: A Language for High-Level Configuration Management“. In: *Proceedings of the 21th Large Installation System Administration Conference (LISA 2007)*. Usenix, Nov. 2007, S. 261–273 (Zitiert auf S. 42, 63).
- [DJV10] T. Delaet, W. Joosen und B. Vanbrabant. „A Survey of System Configuration Tools“. In: *Proceedings of the 24th International Conference on Large Installation System Administration (LISA 2010)*. USENIX, Nov. 2010 (Zitiert auf S. 39).
- [DK75] F. DeRemer und H. Kron. „Programming-in-the Large versus Programming-in-the-small“. In: *Proceedings of the International Conference on Reliable Software*. ACM, 1975, S. 114–121 (Zitiert auf S. 74).
- [DLT+14] R. Di Cosmo, M. Lienhardt, R. Treinen, S. Zacchiroli, J. Zwolakowski, A. Eiche und A. Agahi. „Automated Synthesis and Deployment of Cloud Applications“. In: *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE 2014)*. ACM, Sep. 2014, S. 211–222 (Zitiert auf S. 61).
- [DMT09] DMTF. *Open Virtualization Format (OVF)*. Distributed Management Task Force (DMTF). 2009 (Zitiert auf S. 33).
- [DMT10] DMTF. *Common Information Model (CIM)*. Distributed Management Task Force (DMTF). 2010 (Zitiert auf S. 29, 73, 164).
- [DMT12] DMTF. *Cloud Infrastructure Management Interface (CIMI) Version 1.0.0*. Distributed Management Task Force (DMTF). 2012 (Zitiert auf S. 50, 51).

- [DMZZ14] R. Di Cosmo, J. Mauro, S. Zacchiroli und G. Zavattaro. „Aeolus: a Component Model for the Cloud“. In: *Information and Computation* (Jan. 2014), S. 100–121 (Zitiert auf S. 60).
- [DRDN07] T. Dumitraş, D. Roşu, A. Dan und P. Narasimhan. „Ecotopia: An Ecological Framework for Change Management in Distributed Systems“. In: *Architecting Dependable Systems IV*. Bd. 4615. Springer, 2007, S. 262–286 (Zitiert auf S. 85).
- [dSdL11] C. E. da Silva und R. de Lemos. „A Framework for Automatic Generation of Processes for Self-Adaptive Software Systems“. In: *Informatica* 35.1 (2011), S. 3–13 (Zitiert auf S. 235).
- [DVdJ04] E. Dolstra, E. Visser und M. de Jonge. „Imposing a Memory Management Discipline on Software Deployment“. In: *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*. IEEE, Mai 2004, S. 583–592 (Zitiert auf S. 55).
- [DVL13] E. Dolstra, R. Vermaas und S. Levy. „Charon: Declarative Provisioning and Deployment“. In: *Proceedings of the 1st International Workshop on Release Engineering (RELENG 2013)*. Mai 2013, S. 17–20 (Zitiert auf S. 55).
- [DZZ12] R. Di Cosmo, S. Zacchiroli und G. Zavattaro. „Towards a Formal Component Model for the Cloud“. In: *Software Engineering and Formal Methods*. Springer, 2012, S. 156–171 (Zitiert auf S. 60).
- [Ecl15] Eclipse. *BPEL Designer Project*. Mai 2015. URL: <https://www.eclipse.org/bpel/> (Zitiert auf S. 229).
- [EEKS11] T. Eilam, M. Elder, A. Konstantinou und E. Snible. „Pattern-based Composite Application Deployment“. In: *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*. IEEE, Mai 2011, S. 217–224 (Zitiert auf S. 61, 62, 233, 236).
- [EKK+06] T. Eilam, M. Kalantar, A. Konstantinou, G. Pacifici, J. Pershing und A. Agrawal. „Managing the configuration complexity of distributed applications in Internet data centers“. In: *Communications Magazine* 44.3 (März 2006), S. 166–177 (Zitiert auf S. 46, 68, 268).

- [EKK09] T. Eilam, M. Kalantar und A. Konstantinou. *IBM Research Report: Distributed and Collaborative Design of Composite Service Deployments Using Graph Transformations*. Techn. Ber. IBM Research Division, 2009, S. 1–13 (Zitiert auf S. 268).
- [EKKP05] T. Eilam, M. Kalantar, A. Konstantinou und G. Pacifici. „Reducing the complexity of application deployment in large data centers“. In: *Proceedings of the 9th International Symposium on Integrated Network Management (IM 2005)*. IEEE, Mai 2005, S. 221–234 (Zitiert auf S. 268).
- [EME+06] K. El Maghraoui, A. Meghranjani, T. Eilam, M. Kalantar und A. Konstantinou. „Model Driven Provisioning: Bridging the Gap Between Declarative Object Models and Procedural Provisioning Tools“. In: *Proceedings of the 7th International Middleware Conference (Middleware 2006)*. Springer, Nov. 2006, S. 404–423 (Zitiert auf S. 41, 42, 56, 57, 75, 76, 235, 293).
- [EN10] R. Elmasri und S. B. Navathe. *Fundamentals of Database Systems (6th Edition)*. Pearson, Apr. 2010 (Zitiert auf S. 51).
- [Ern10] A. M. Ernst. „A Pattern-based Approach to Enterprise Architecture Management“. Dissertation. Technische Universität München, 2010 (Zitiert auf S. 90).
- [EUL09] H. Eberle, T. Unger und F. Leymann. „Process Fragments“. In: *On the Move to Meaningful Internet Systems: OTM 2009*. Springer, 2009, S. 398–405 (Zitiert auf S. 237).
- [EYG97] A. Eden, A. Yehudai und J. Gil. „Precise Specification and Automatic Application of Design Patterns“. In: *Proceedings of the 12th IEEE International Conference Automated Software Engineering (ASE 1997)*. IEEE, Nov. 1997, S. 143–152 (Zitiert auf S. 269).
- [FBB+14a] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling und F. Leymann. „Efficient Pattern Application: Validating the Concept of Solution Implementations in Different Domains“. In: *International Journal On Advances in Software 7.3&4* (Dez. 2014), S. 710–726 (Zitiert auf S. 94, 232, 296).
- [FBB+14b] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling und F. Leymann. „From Pattern Languages to Solution Implementations“. In: *Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014)*. Xpert Publishing Services, Mai 2014, S. 12–21 (Zitiert auf S. 93, 94, 232, 296).

- [FBB+15] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, F. Leymann, A. Hadjakos, F. Hentschel und H. Schulze. „Leveraging Pattern Application via Pattern Refinement“. In: *Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC 2015)*. epubli, Juni 2015 (Zitiert auf S. 90).
- [FBBL14] C. Fehling, J. Barzen, U. Breitenbücher und F. Leymann. „A Process for Pattern Identification, Authoring, and Application“. In: *Proceedings of the 19th European Conference on Pattern Languages of Programs (EuroPLOP 2014)*. ACM, Jan. 2014 (Zitiert auf S. 91).
- [FBFL15] C. Fehling, J. Barzen, M. Falkenthal und F. Leymann. „PatternPedia – Collaborative Pattern Identification and Authoring“. In: *Proceedings of PURPLSOC (Pursuit of Pattern Languages for Societal Change). The Workshop 2014*. Aug. 2015, S. 252–284 (Zitiert auf S. 91).
- [FBR+14] N. Ferry, G. Brataas, A. Rossini, F. Chauvel und A. Solberg. „Towards Bridging the Gap Between Scalability and Elasticity“. In: *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014)*. SciTePress, Apr. 2014, S. 746–751 (Zitiert auf S. 33).
- [FCR+13] N. Ferry, F. Chauvel, A. Rossini, B. Morin und A. Solberg. „Managing Multi-cloud Systems with CloudMF“. In: *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies (NordiCloud 2013)*. ACM, Sep. 2013, S. 38–45 (Zitiert auf S. 33).
- [Fei15] L. Feinberg. *715 hours of downtime across 51 major events in 2014*. Jan. 2015. URL: <https://www.cloudendure.com/blog/715-hours-downtime-across-51-major-events-2014/> (Zitiert auf S. 13, 41).
- [FEL+12] C. Fehling, T. Ewald, F. Leymann, M. Pauly, J. Rütschlin und D. Schumm. „Capturing Cloud Computing Knowledge and Experience in Patterns“. In: *Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD 2012)*. IEEE, Juni 2012, S. 726–733 (Zitiert auf S. 91).
- [FH10] S. Frey und W. Hasselbring. „Model-Based Migration of Legacy Software Systems to Scalable and Resource-Efficient Cloud-Based Applications: The CloudMIG Approach“. In: *Proceedings of the First International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2010)*. IARIA, Nov. 2010, S. 155–158 (Zitiert auf S. 36).

- [FH11a] S. Frey und W. Hasselbring. „An Extensible Architecture for Detecting Violations of a Cloud Environment’s Constraints during Legacy Software System Migration“. In: *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR 2011)*. IEEE, März 2011, S. 269–278 (Zitiert auf S. 36).
- [FH11b] S. Frey und W. Hasselbring. „The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications“. In: *International Journal On Advances in Software* 4.3&4 (2011), S. 342–353 (Zitiert auf S. 36).
- [FHK+09] U. Frank, D. Heise, H. Kattenstroth, D. Ferguson, E. Hadar und M. Waschke. „ITML: A Domain-Specific Modeling Language for Supporting Business Driven IT Management“. In: *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM 2009)*. Helsinki Business School, Okt. 2009 (Zitiert auf S. 28, 36, 40).
- [Fie00] R. T. Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Dissertation. University of California, Irvine, 2000 (Zitiert auf S. 50).
- [FL14] C. Fehling und F. Leymann. *PatternPedia: a Wiki for Patterns*. Technischer Bericht Informatik 2014/03. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Juni 2014, S. 8 (Zitiert auf S. 91).
- [FLR+11] C. Fehling, F. Leymann, R. Retter, D. Schumm und W. Schupeck. „An Architectural Pattern Language of Cloud-based Applications“. In: *Proceedings of the 18th Conference on Pattern Languages of Programs (PLOP 2011)*. ACM, Okt. 2011 (Zitiert auf S. 91, 97).
- [FLR+13] C. Fehling, F. Leymann, S. T. Ruehl, M. Rudek und S. Verclas. „Service Migration Patterns - Decision Support and Best Practices for the Migration of Existing Service-based Applications to Cloud Environments“. In: *Proceedings of the 6th IEEE International Conference on Service Oriented Computing and Applications (SOCA 2013)*. IEEE, Dez. 2013, S. 9–16 (Zitiert auf S. 13, 20, 41, 92, 93, 104, 242).

- [FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck und P. Arbitter. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer, Jan. 2014, S. 367 (Zitiert auf S. [13](#), [20](#), [41](#), [90–93](#), [242](#)).
- [FLRS12] C. Fehling, F. Leymann, J. Rütschlin und D. Schumm. „Pattern-Based Development and Management of Cloud Applications“. In: *Future Internet* 4.1 (März 2012), S. 110–141 (Zitiert auf S. [41](#), [42](#), [44](#), [92](#), [93](#), [95](#), [242](#)).
- [Fow02] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, Nov. 2002 (Zitiert auf S. [13](#), [26](#), [90](#), [91](#)).
- [FRC+13] N. Ferry, A. Rossini, F. Chauvel, B. Morin und A. Solberg. „Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems“. In: *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD 2013)*. IEEE, Juli 2013, S. 887–894 (Zitiert auf S. [32](#), [42](#)).
- [FS06] D. F. Ferguson und M. L. Stockton. „Enterprise Business Process Management: Architecture, Technology and Standards“. In: *Proceedings of the 4th International Conference on Business Process Management (BPM 2006)*. Springer, Sep. 2006, S. 1–15 (Zitiert auf S. [31](#)).
- [FSR+14] N. Ferry, H. Song, A. Rossini, F. Chauvel und A. Solberg. „CloudMF: Applying MDE to Tame the Complexity of Managing Multi-Cloud Applications“. In: *Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2014)*. IEEE, Dez. 2014, S. 269–277 (Zitiert auf S. [32](#), [33](#), [36](#)).
- [FTLW14] M. Fleck, J. Troya, P. Langer und M. Wimmer. „Towards Pattern-Based Optimization of Cloud Applications“. In: *Proceedings of the 2nd International Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE 2014)*. CEUR-WS.org, Sep. 2014, S. 16–25 (Zitiert auf S. [96](#)).
- [FZBX14] M. Fu, L. Zhu, L. Bass und X. Xu. „A Recoverability-Oriented Analysis for Operations on Cloud Applications“. In: *Proceedings of the 2014 IEEE/IFIP Conference on Software Architecture (WICSA 2014)*. IEEE, Apr. 2014, S. 125–128 (Zitiert auf S. [84](#)).

- [Gad14] O. Gadish. *Top 9 Reasons for Cloud Application Failure*. Aug. 2014. URL: <https://www.cloudendure.com/blog/top-9-reasons-cloud-application-failure/> (Zitiert auf S. 13, 41).
- [GES+11] G. Gonçalves, P. Endo, M. Santos, D. Sadok, J. Kelner, B. Melander und J.-E. Mangs. „CloudML: An Integrated Language for Resource, Service and Request Description for D-Clouds“. In: *Proceedings of the Third International Conference on Cloud Computing Technology and Science (CloudCom 2011)*. IEEE, Nov. 2011, S. 399–406 (Zitiert auf S. 33).
- [GFJK03] P. Greenfield, K. Fekete, J. Jang und D. Kuo. „Compensation is Not Enough“. In: *Proceedings of the Seventh IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003)*. IEEE, Sep. 2003, S. 232–239 (Zitiert auf S. 233).
- [GGL+09] P. Goldsack, J. Guijarro, S. Loughran, A. Coles, A. Farrell, A. Lain, P. Murray und P. Toft. „The SmartFrog Configuration Management Framework“. In: *SIGOPS Operating Systems Review* 43.1 (Jan. 2009), S. 16–25 (Zitiert auf S. 55).
- [GHJV94] E. Gamma, R. Helm, R. Johnson und J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, Okt. 1994 (Zitiert auf S. 13, 91, 96).
- [GJ79] M. R. Garey und D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, Apr. 1979 (Zitiert auf S. 62).
- [GMMC13a] J. Guillén, J. Miranda, J. M. Murillo und C. Canal. „A service-oriented framework for developing cross cloud migratable software“. In: *The Journal of Systems and Software* 86.9 (Sep. 2013), S. 2294–2308 (Zitiert auf S. 35).
- [GMMC13b] J. Guillén, J. Miranda, J. Murillo und C. Canal. „A UML Profile for Modeling Multicloud Applications“. In: *Proceedings of the Second European Conference on Service-Oriented and Cloud Computing (ESOCC 2013)*. Springer, Sep. 2013, S. 180–187 (Zitiert auf S. 35).
- [GMW00] D. Garlan, R. T. Monroe und D. Wile. „Acme: Architectural Description of Component-Based Systems“. In: *Foundations of Component-Based Systems*. Cambridge University Press, 2000, S. 47–68 (Zitiert auf S. 26).

- [GMW97] D. Garlan, R. Monroe und D. Wile. „ACME: An Architecture Description Interchange Language“. In: *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON 1997)*. IBM Press, Nov. 1997, S. 169–183 (Zitiert auf S. 26).
- [GNT04] M. Ghallab, D. Nau und P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann, Mai 2004 (Zitiert auf S. 80).
- [GP95] D. Garlan und D. E. Perry. „Introduction to the Special Issue on Software Architecture“. In: *IEEE Transactions on Software Engineering* 21.4 (Apr. 1995), S. 269–274 (Zitiert auf S. 26).
- [Gra10] G. Graefe. „A Survey of B-tree Locking Techniques“. In: *ACM Transactions on Database Systems* 35.3 (Juli 2010), 16:1–16:26 (Zitiert auf S. 87).
- [Gra86] J. Gray. „Why Do Computers Stop and What Can Be Done About It?“. In: *5th Symposium on Reliability in Distributed Software and Database Systems*. 1986, S. 3–12 (Zitiert auf S. 27).
- [GS87] H. Garcia-Molina und K. Salem. „Sagas“. In: *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference (SIGMOD 1987)*. ACM, Mai 1987, S. 249–259 (Zitiert auf S. 227).
- [GSR+09] F. Galán, A. Sampaio, L. Roderio-Merino, I. Loy, V. Gil und L. M. Vaquero. „Service Specification in Cloud Environments Based on Extensions to Open Standards“. In: *Proceedings of the 4th International Conference on COMMunication System softWARE and MiddlewaRE (COMSWARE 2009)*. ACM, Juni 2009 (Zitiert auf S. 33).
- [HA11] J. A. Hewson und P. Anderson. „Modelling System Administration Problems with CSPs“. In: *Proceedings of the 10th International Workshop on Constraint Modelling and Reformulation (ModRef 2011)*. 2011, S. 73–82 (Zitiert auf S. 53).
- [HA12] H. Herry und P. Anderson. „Planning with Global Constraints for Computing Infrastructure Reconfiguration“. In: *Proceedings of the the AAAI Workshop on Problem Solving using Classical Planners (CP4PS 2012)*. AAAI, 2012, S. 44–50 (Zitiert auf S. 58).

- [HAG12] J. A. Hewson, P. Anderson und A. D. Gordon. „A Declarative Approach to Automated Configuration“. In: *Proceedings of the 26th Large Installation System Administration Conference (LISA 12)*. USENIX, Dez. 2012, S. 51–66 (Zitiert auf S. 53, 84).
- [HAR13] H. Herry, P. Anderson und M. Rovatsos. „Choreographing Configuration Changes“. In: *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*. IEEE, Okt. 2013, S. 156–160 (Zitiert auf S. 59).
- [Har99] N. Harrison. „The Language of Shepherds: A Pattern Language for Shepherding“. In: *Proceedings of the 6th Annual Conference on the Pattern Languages of Programs (PLoP 1999)*. Aug. 1999 (Zitiert auf S. 91).
- [HAW11] H. Herry, P. Anderson und G. Wickler. „Automated Planning for Configuration Changes“. In: *Proceedings of the 25th International Conference on Large Installation System Administration (LISA 2011)*. USENIX, Dez. 2011, S. 57–68 (Zitiert auf S. 54, 58, 65, 68, 69, 88, 94).
- [HAZ07] N. Harrison, P. Avgeriou und U. Zdun. „Using Patterns to Capture Architectural Decisions“. In: *Software* 24.4 (Juli 2007), S. 38–45 (Zitiert auf S. 91).
- [HdCG+12] S. Hagen, W. L. da Costa Cordeiro, L. P. Gasparly, L. Z. Granville, M. Seibold und A. Kemper. „Planning in the Large: Efficient Generation of IT Change Plans on Large Infrastructures“. In: *Proceedings of the 8th International Conference on Network and Service Management (CNSM 2012)*. IEEE, Okt. 2012, S. 108–116 (Zitiert auf S. 81).
- [HdCG+14] S. Hagen, W. L. da Costa Cordeiro, L. P. Gasparly, L. Zambenedetti Granville und A. Kemper. „Efficient Model Checking of IT Change Operations“. In: *IEEE Transactions on Network and Service Management* 11.3 (Sep. 2014), S. 292–306 (Zitiert auf S. 41, 84).
- [HEW+09] S. Hagen, N. Edwards, L. Wilcock, J. Kirschnick und J. Rolia. „One Is Not Enough: A Hybrid Approach for IT Change Planning“. In: *Integrated Management of Systems, Services, Processes and People in IT*. Springer, Okt. 2009, S. 56–70 (Zitiert auf S. 42, 81).
- [Hew13] J. A. Hewson. „Constraint-based specifications for system configuration“. Dissertation. The University of Edinburgh, 2013 (Zitiert auf S. 39).

- [HF10] J. Humble und D. Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley, Juli 2010 (Zitiert auf S. 39, 41–44).
- [HFK+14] F. Haupt, M. Fischer, D. Karastoyanova, F. Leymann und K. Vukojevic-Haupt. „Service Composition for REST“. In: *Proceedings of the 18th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2014)*. IEEE, Sep. 2014, S. 110–119 (Zitiert auf S. 230, 294).
- [HK10a] S. Hagen und A. Kemper. „Model-Based Planning for State-Related Changes to Infrastructure and Software as a Service Instances in Large Data Centers“. In: *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD 2010)*. IEEE, Juli 2010, S. 11–18 (Zitiert auf S. 60).
- [HK10b] S. Hagen und A. Kemper. „Facing the Unpredictable: Automated Adaptation of IT Change Plans for Unpredictable Management Domains“. In: *Proceedings of the 2010 International Conference on Network and Service Management (CNSM 2010)*. Okt. 2010, S. 33–40 (Zitiert auf S. 29, 78, 105, 234).
- [HK11] S. Hagen und A. Kemper. „Towards Solid IT Change Management: Automated Detection of Conflicting IT Change Plans“. In: *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*. IEEE, 2011, S. 265–272 (Zitiert auf S. 84, 85).
- [HP12] The Open Group. *Archimate 2.0 Specification*. Van Haren Publishing, Feb. 2012 (Zitiert auf S. 26, 36, 164).
- [HSK12] S. Hagen, M. Seibold und A. Kemper. „Efficient Verification of IT Change Operations or: How We Could Have Prevented Amazon’s Cloud Outage“. In: *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2012)*. IEEE, Apr. 2012, S. 368–376 (Zitiert auf S. 84).
- [Hud98] P. Hudak. „Modular domain specific languages and tools“. In: *Proceedings of the Fifth International Conference on Software Reuse (ICSR 1998)*. IEEE, Juni 1998, S. 134–142 (Zitiert auf S. 31, 121).
- [Hüt12] M. Hüttermann. *DevOps for Developers*. Apress, Sep. 2012 (Zitiert auf S. 39, 52).

- [HW04] G. Hohpe und B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2004 (Zitiert auf S. 13, 91, 96, 290).
- [HZ15] T. Haitzer und U. Zdun. „Semi-automatic architectural pattern identification and documentation using architectural primitives“. In: *Journal of Systems and Software* 102 (2015), S. 35–57 (Zitiert auf S. 269).
- [HZR10] S. Herden, A. Zwanziger und P. Robinson. „Declarative Application Deployment and Change Management“. In: *Proceedings of the 2010 International Conference on Network and Service Management (CNSM 2010)*. IEEE, Okt. 2010, S. 126–133 (Zitiert auf S. 43, 55, 266).
- [IEE00] IEEE Standards Association. *IEEE Std 1471 (2000): IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*. 2000 (Zitiert auf S. 26).
- [IEE90] IEEE Standards Association. *IEEE Std 610.12 (1990): IEEE Standard Glossary of Software Engineering Terminology*. 1990 (Zitiert auf S. 26, 39).
- [ISA16] ISACA. *COBIT 5: A Business Framework for the Governance and Management of Enterprise IT*. Feb. 2016. URL: <http://www.isaca.org/cobit/pages/default.aspx> (Zitiert auf S. 40).
- [JK05] F. Jouault und I. Kurtev. „Transforming Models with ATL“. In: *Proceedings of the 2005 International Conference on Satellite Events at the MoDELS (MoDELS 2005)*. Springer, Okt. 2005, S. 128–138 (Zitiert auf S. 252).
- [JPCL14] P. Jamshidi, C. Pahl, S. Chinenyeze und X. Liu. „Cloud Migration Patterns: A Multi-Cloud Service Architecture Perspective“. In: *Service-Oriented Computing - ICSOC 2014 Workshops*. Springer, Nov. 2014, S. 6–19 (Zitiert auf S. 13, 20, 92, 266).
- [JRVL00] M. Jazayeri, A. Ran und F. van der Linden. *Software Architecture for Product Families: Principles and Practice*. Addison-Wesley, Jan. 2000 (Zitiert auf S. 26).
- [KA08] A. W. Kamal und P. Avgeriou. „Modeling Architectural Patterns' Behavior Using Architectural Primitives“. In: *Proceedings of the Second European Conference on Software Architecture (ECSA 2008)*. Springer, Okt. 2008, S. 164–179 (Zitiert auf S. 259).

- [KAG+12] J. Kirschnick, J. M. Alcaraz Calero, P. Goldsack, A. Farrell, J. Guijarro, S. Loughran, N. Edwards und L. Wilcock. „Towards an Architecture for Deploying Elastic Services in the Cloud“. In: *Software: Practice and Experience* 42.4 (Apr. 2012), S. 395–408 (Zitiert auf S. 33).
- [KB04] A. Keller und R. Badonnel. „Automating the Provisioning of Application Services with the BPEL4WS Workflow Language“. In: *Proceedings of the 15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2004)*. Springer, Nov. 2004, S. 15–27 (Zitiert auf S. 70, 80).
- [KBBL12] O. Kopp, T. Binz, U. Breitenbücher und F. Leymann. „BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications“. In: *Proceedings of the 4th International Workshop on the Business Process Model and Notation (BPMN 2012)*. Springer, Sep. 2012, S. 38–52 (Zitiert auf S. 70, 72).
- [KBBL13] O. Kopp, T. Binz, U. Breitenbücher und F. Leymann. „Winery – A Modeling Tool for TOSCA-based Cloud Applications“. In: *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*. Springer, Dez. 2013, S. 700–704 (Zitiert auf S. 73, 295).
- [KBH07] A. Keller, A. Brown und J. Hellerstein. „A Configuration Complexity Model and Its Application to a Change Management System“. In: *Transactions on Network and Service Management* 4.1 (Juni 2007), S. 13–27 (Zitiert auf S. 42).
- [KBK00] A. Keller, U. Blumenthal und G. Kar. „Classification and Computation of Dependencies for Distributed Management“. In: *Proceedings of the Fifth IEEE Symposium on Computers and Communications (ISCC 2000)*. IEEE, Juli 2000, S. 78–83 (Zitiert auf S. 29, 82).
- [KBS04] D. Krafzig, K. Banke und D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall PTR, Nov. 2004 (Zitiert auf S. 26).
- [KEK+09] A. V. Konstantinou, T. Eilam, M. Kalantar, A. A. Totok, W. Arnold und E. Snible. „An Architecture for Virtual Solution Composition and Deployment in Infrastructure Clouds“. In: *Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing (VTDC 2009)*. ACM, Juni 2009, S. 9–18 (Zitiert auf S. 33).

- [KELU10] O. Kopp, H. Eberle, F. Leymann und T. Unger. „The Subprocess Spectrum“. In: *Proceedings of the Business Process and Services Computing Conference (BPSC 2010)*. Bd. P-177. Gesellschaft für Informatik e.V. (GI), Sep. 2010, S. 267–279 (Zitiert auf S. [71](#), [169](#), [225](#), [226](#)).
- [KG14a] T. Kuroda und A. Gokhale. „Model-based Automation for Hardware Provisioning in IT Infrastructure“. In: *Proceedings of the 8th Annual IEEE Systems Conference (SysCon 2014)*. IEEE, März 2014, S. 293–300 (Zitiert auf S. [61](#)).
- [KG14b] T. Kuroda und A. Gokhale. „Model-Based IT Change Management for Large System Definitions with State-Related Dependencies“. In: *Proceedings of the 18th International Conference on Enterprise Distributed Object Computing (EDOC 2014)*. IEEE, Sep. 2014, S. 170–179 (Zitiert auf S. [61](#)).
- [KHW+04] A. Keller, J. L. Hellerstein, J. L. Wolf, K.-L. Wu und V. Krishnan. „The CHAMPS System: Change Management with Planning and Scheduling“. In: *Proceedings of the 10th Network Operations and Management Symposium (NOMS 2004)*. IEEE, Apr. 2004, S. 395–408 (Zitiert auf S. [70](#), [79](#)).
- [KK01] A. Keller und G. Kar. „Determining Service Dependencies in Distributed Systems“. In: *Proceedings of the International Conference on Communications (ICC 2001)*. IEEE, Juni 2001, S. 2084–2088 (Zitiert auf S. [29](#)).
- [KKL+05] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt und I. Trickovic. „WS-BPEL Extensions for Sub-processes–BPEL-SPE“. In: *SAP/IBM*, Sep. 2005 (Zitiert auf S. [71](#), [173](#), [226](#), [237](#)).
- [KKL08] O. Kopp, R. Khalaf und F. Leymann. „Deriving Explicit Data Links in WS-BPEL Processes“. In: *Proceedings of the International Conference on Services Computing (SCC 2008)*. IEEE, Juli 2008, S. 367–376 (Zitiert auf S. [70](#)).
- [Kle13] F. Kleiner. *IT Service Management: Aus der Praxis für die Praxis*. Springer, Juni 2013 (Zitiert auf S. [40](#)).
- [KMWL09] O. Kopp, D. Martin, D. Wutke und F. Leymann. „The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages“. In: *Enterprise Modelling and Information Systems 4.1* (Juni 2009), S. 3–13 (Zitiert auf S. [225](#)).

- [Koo06] Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt). *ITIL und Standards für IT-Prozesse - Version 1.0.1*. Techn. Ber. 2006 (Zitiert auf S. 40).
- [Kow79] R. Kowalski. „Algorithm = Logic + Control“. In: *Communications of the ACM* 22.7 (Juli 1979), S. 424–436 (Zitiert auf S. 51).
- [KS09] A. Keller und S. Subramanian. „Best Practices for Deploying a CMDB in Large-scale Environments“. In: *Proceedings of the 11th IFIP/IEEE International Symposium on Integrated Network Management (IM 2009)*. IEEE, Juni 2009, S. 732–745 (Zitiert auf S. 29).
- [KVC09] G. Koslovski, P. Vicat-Blanc Primet und A. S. Charão. „VXDL: Virtual Resources and Interconnection Networks Description Language“. In: *Networks for Grid Applications (GridNets 2009)*. Springer, Sep. 2009, S. 138–154 (Zitiert auf S. 33).
- [KW03] F. Keller und S. Wendt. „FMC: An Approach Towards Architecture-centric System Development“. In: *Proceedings of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*. IEEE, Apr. 2003, S. 173–182 (Zitiert auf S. 26, 36, 164).
- [LA94] F. Leymann und W. Altenhuber. „Managing business processes as an information resource“. In: *IBM Systems Journal* 33.2 (1994), S. 326–348 (Zitiert auf S. 216).
- [LC01] B. Leuf und W. Cunningham. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley, Mai 2001 (Zitiert auf S. 91).
- [LdAA+04] D. Lucrédio, E. S. de Almeida, A. Alvaro, V. C. Garcia und E. K. Piveta. „Student’s PLoP Guide: A Pattern Family to Guide Computer Science Students during PLoP Conferences“. In: *Proceedings of the 4th Latin American Conference on Patterns Languages of Programming (SugarLoafPlop 2004)*. 2004 (Zitiert auf S. 91).
- [Ley10] F. Leymann. „BPEL vs. BPMN 2.0: Should You Care?“ In: *Business Process Modeling Notation*. Springer, Okt. 2010, S. 8–13 (Zitiert auf S. 37, 70).
- [Ley95] F. Leymann. „Supporting Business Transactions via Partial Backward Recovery In Workflow Management Systems“. In: *Datenbanksysteme in Büro, Technik und Wissenschaft*. Informatik aktuell. Springer, 1995, S. 51–70 (Zitiert auf S. 227).

- [LFM+11] F. Leymann, C. Fehling, R. Mietzner, A. Nowak und S. Dustdar. „Moving Applications to the Cloud: An Approach based on Application Model Enrichment“. In: *International Journal of Cooperative Information Systems* 20.3 (Sep. 2011), S. 307–356 (Zitiert auf S. 31).
- [Lie06] G. Lienemann. *ITIL - Change Management: Hinweise und Vorgehensweisen aus der Praxis*. Heise Medien, Mai 2006 (Zitiert auf S. 39, 40).
- [LL07] J. Ludewig und H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. dpunkt.verlag GmbH, 2007 (Zitiert auf S. 26, 38, 39, 105).
- [Llo94] J. W. Lloyd. „Practical Advantages of Declarative Programming“. In: *Proceedings of the Joint Conference on Declarative Programming*. 1994, S. 3–17 (Zitiert auf S. 51, 52).
- [LLQ07] J. Luo, Y. Li und J. Qiu. „Pattern based Declarative Constraint Solution for Model-Driven Service Deployment“. In: *Proceedings of the IEEE International Conference on Services Computing (SCC 2007)*. IEEE, Juli 2007, S. 711–712 (Zitiert auf S. 284).
- [LLQC08] J. Luo, Y. Li, J. Qiu und Y. Chen. „Declarative Constraint Framework for SOA Deployment and Configuration“. In: *Proceedings of the IEEE International Conference on Web Services (ICWS 2008)*. IEEE, Sep. 2008, S. 637–644 (Zitiert auf S. 284).
- [LMZ13] T. Lascu, J. Mauro und G. Zavattaro. „A Planning Tool Supporting the Deployment of Cloud Applications“. In: *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2013)*. IEEE, Nov. 2013, S. 213–220 (Zitiert auf S. 61).
- [LMZ15] T. A. Lascu, J. Mauro und G. Zavattaro. „Automatic Deployment of Component-based Applications“. In: *Science of Computer Programming* 113.3 (2015), S. 261–284 (Zitiert auf S. 61).
- [LO04] J. Ludewig und S. Opferkuch. „Software-Wartung - Eine Taxonomie“. In: *Softwaretechnik-Trends* 24.2 (Mai 2004), S. 35–36 (Zitiert auf S. 39).
- [LR00] F. Leymann und D. Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 2000 (Zitiert auf S. 67, 70, 71, 109, 116, 225–227, 237).

- [LR09] K. Levanti und A. Ranganathan. „Planning-based Configuration and Management of Distributed Systems“. In: *Proceedings of the 11th IFIP/IEEE International Symposium on Integrated Network Management (IM 2009)*. IEEE, Juni 2009, S. 65–72 (Zitiert auf S. [59](#), [63](#), [236](#)).
- [LSS+13] H. Lu, M. Shtern, B. Simmons, M. Smit und M. Litoiu. „Pattern-Based Deployment Service for Next Generation Clouds“. In: *Proceedings of the IEEE Ninth World Congress on Services (SERVICES 2013)*. IEEE, Juni 2013, S. 464–471 (Zitiert auf S. [33](#)).
- [LTC+14] D.-H. Le, H.-L. Truong, G. Copil, S. Nastic und S. Dustdar. „SALSA: A Framework for Dynamic Configuration of Cloud Services“. In: *Proceedings of the 6th International Conference on Cloud Computing Technology and Science (CloudCom 2014)*. IEEE, Dez. 2014, S. 146–153 (Zitiert auf S. [13](#)).
- [LZ11] D. Liu und J. Zic. „Cloud#: A Specification Language for Modeling Cloud“. In: *Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD 2011)*. IEEE, Juli 2011, S. 533–540 (Zitiert auf S. [33](#)).
- [MBJ+09] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey und A. Solberg. „Models@Run.Time to Support Dynamic Adaptation“. In: *Computer* 42.10 (Okt. 2009), S. 44–51 (Zitiert auf S. [55](#)).
- [MD97] G. Meszaros und J. Doble. „Pattern Languages of Program Design 3“. In: Addison-Wesley, 1997. Kap. A Pattern Language for Pattern Writing, S. 529–574 (Zitiert auf S. [91](#)).
- [MdCdS+09] G. S. Machado, W. L. da Costa Cordeiro, A. D. dos Santos, J. Wickboldt, R. C. Lunardi, F. G. Andreis, C. B. Both, L. P. Gaspary, L. Z. Granville, D. Trastour und C. Bartolini. „Refined Failure Remediation for IT Change Management Systems“. In: *Proceedings of the 11th IFIP/IEEE International Symposium on Integrated Network Management (IM 2009)*. IEEE, Juni 2009, S. 638–645 (Zitiert auf S. [85](#), [233](#)).
- [MDdC+08] G. S. Machado, F. F. Daitx, W. L. da Costa Cordeiro, C. B. Both, L. P. Gaspary, L. Z. Granville, C. Bartolini, A. Sahai, D. Trastour und K. Srikoski. „Enabling Rollback Support in IT Change Management Systems“. In: *Proceedings of the IEEE/IFIP Network Operations and Management*

- Symposium (NOMS 2008)*. IEEE, Apr. 2008, S. 347–354 (Zitiert auf S. [85](#), [233](#)).
- [MDW+00] V. Machiraju, M. Dekhil, K. Wurster, P. K. Garg, M. L. Griss und J. Holland. „Towards Generic Application Auto-Discovery“. In: *Proceedings of the 7th IEEE/IFIP Network Operations and Management Symposium (NOMS 2000)*. IEEE, Apr. 2000, S. 75–87 (Zitiert auf S. [29](#)).
- [MGH+98] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld und D. Wilkins. *PDDL - The Planning Domain Definition Language*. Techn. Ber. Yale Center for Computational Vision und Control, Okt. 1998 (Zitiert auf S. [57](#)).
- [MHS05] M. Mernik, J. Heering und A. M. Sloane. „When and How to Develop Domain-specific Languages“. In: *ACM Computing Surveys* 37.4 (Dez. 2005), S. 316–344 (Zitiert auf S. [31](#), [121](#), [122](#)).
- [Mie10] R. Mietzner. „A Method and Implementation to Define and Provision Variable Composite Applications, and its Usage in Cloud Computing“. Dissertation. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2010 (Zitiert auf S. [27](#), [31](#), [70](#)).
- [ML08] R. Mietzner und F. Leymann. „Towards Provisioning the Cloud: On the Usage of Multi-Granularity Flows and Services to Realize a Unified Provisioning Infrastructure for SaaS Applications“. In: *Proceedings of the International Congress on Services (SERVICES 2008)*. IEEE, Juli 2008, S. 3–10 (Zitiert auf S. [232](#), [236](#)).
- [MM03] N. R. Mehta und N. Medvidovic. „Composing Architectural Styles from Architectural Primitives“. In: *Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2011)*. ACM, Sep. 2003, S. 347–350 (Zitiert auf S. [259](#)).
- [Moo09] D. Moody. „The Physics of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering“. In: *IEEE Transactions on Software Engineering* 35.6 (Nov. 2009), S. 756–779 (Zitiert auf S. [37](#), [155](#), [160](#)).
- [MR14] M. Mohaan und R. Raithatha. *Learning Ansible*. Packt Publishing, Nov. 2014 (Zitiert auf S. [52](#)).

- [MRM12] A. Marrella, A. Russo und M. Mecella. „Planlets: Automatically Recovering Dynamic Processes in YAWL“. In: *On the Move to Meaningful Internet Systems (OTM 2012)*. Springer, Sep. 2012, S. 268–286 (Zitiert auf S. 168).
- [MSB+07] H. Madduri, S. S. Shi, R. Baker, N. Ayachitula, L. Schwartz, M. Surendra, C. Corley, M. Benantar und S. Patel. „A Configuration Management Database Architecture in Support of IBM Service Management“. In: *IBM Systems Journal* 46.3 (März 2007), S. 441–457 (Zitiert auf S. 29).
- [MUL09] R. Mietzner, T. Unger und F. Leymann. „Cafe: A Generic Configurable Customizable Composite Cloud Application Framework“. In: *On the Move to Meaningful Internet Systems: OTM 2009 (CoopIS 2009)*. Springer, Nov. 2009, S. 357–364 (Zitiert auf S. 31, 70).
- [NBL14] A. Nowak, U. Breitenbücher und F. Leymann. „Automating Green Patterns to Compensate CO2 Emissions of Cloud-based Business Processes“. In: *Proceedings of the Eighth International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2014)*. Xpert Publishing Services, Aug. 2014, S. 132–139 (Zitiert auf S. 296).
- [Ngu13] D. K. Nguyen. „Blueprint Model and Language for Engineering Cloud Applications“. Dissertation. Tilburg University, 2013 (Zitiert auf S. 35).
- [NL13] A. Nowak und F. Leymann. „Green Business Process Patterns - Part II“. In: *Proceedings of the 6th IEEE International Conference on Service Oriented Computing & Applications (SOCA 2013)*. IEEE, Dez. 2013, S. 168–173 (Zitiert auf S. 91).
- [NLPvH12] D. K. Nguyen, F. Lelli, M. P. Papazoglou und W.-J. van den Heuvel. „Blueprinting Approach in Support of Cloud Computing“. In: *Future Internet* 4.1 (2012), S. 322–346 (Zitiert auf S. 35).
- [NLS+11] A. Nowak, F. Leymann, D. Schleicher, D. Schumm und S. Wagner. „Green Business Process Patterns“. In: *Proceedings of the 18th Conference on Pattern Languages of Programs (PLoP 2011)*. ACM, Okt. 2011 (Zitiert auf S. 91, 242, 296).
- [NLT+11] D. K. Nguyen, F. Lelli, Y. Taher, M. Parkin, M. P. Papazoglou und W.-J. van den Heuvel. „Blueprint Template Support for Engineering Cloud-Based Services“. In: *Towards a Service-Based Internet - 4th European Conference (ServiceWave 2011)*. Springer, Okt. 2011, S. 26–37 (Zitiert auf S. 35).

- [Now14] A. Nowak. „Green Business Process Management: Methode und Realisierung“. Dissertation. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2014 (Zitiert auf S. 91).
- [OAS07a] OASIS. *SCA Assembly Model Specification Version 1.00*. Organization for the Advancement of Structured Information Standards (OASIS). März 2007 (Zitiert auf S. 31).
- [OAS07b] OASIS. *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*. Organization for the Advancement of Structured Information Standards (OASIS). 2007 (Zitiert auf S. 70, 72, 116, 227, 231, 294).
- [OAS08] OASIS. *Open Services for Lifecycle Collaboration (OSLC) Specifications*. Organization for the Advancement of Structured Information Standards (OASIS). 2008 (Zitiert auf S. 40).
- [OAS09] OASIS. *Service Component Architecture WS-BPEL Client and Implementation Specification Version 1.1*. Organization for the Advancement of Structured Information Standards (OASIS). März 2009. URL: <http://docs.oasis-open.org/opencsa/sca-bpel/sca-bpel-1.1-spec.html> (Zitiert auf S. 31).
- [OAS12] OASIS. *Cloud Application Management for Platforms (CAMP) Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS). Aug. 2012 (Zitiert auf S. 51).
- [OAS13a] OASIS. *Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS). 2013 (Zitiert auf S. 27, 32, 73, 153).
- [OAS13b] OASIS. *Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS). 2013 (Zitiert auf S. 27, 32, 71, 73, 121, 163, 168, 294).
- [OAS15] OASIS. *TOSCA Simple Profile in YAML Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS). 2015 (Zitiert auf S. 152, 163).
- [OGF09] OGF. *Open Cloud Computing Interface (OCCI) Specification Version 1.1*. Open Grid Forum (OGF). 2009 (Zitiert auf S. 50).

- [OGP03] D. Oppenheimer, A. Ganapathi und D. A. Patterson. „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 (USITS 2003)*. USENIX, Juni 2003 (Zitiert auf S. 28, 41, 82).
- [OMG07] OMG. *OMG Unified Modeling Language (UML)*. Object Management Group (OMG). 2007 (Zitiert auf S. 29, 164).
- [OMG11a] OMG. *Architecture-Driven Modernization: Knowledge Discovery Meta-Model (KDM) Version 1.3*. Object Management Group (OMG). 2011 (Zitiert auf S. 36).
- [OMG11b] OMG. *Business Process Model and Notation (BPMN) Version 2.0*. Object Management Group (OMG). 2011 (Zitiert auf S. 70).
- [OMG15] OMG. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification (QVT) Version 1.2*. Object Management Group (OMG). Feb. 2015 (Zitiert auf S. 252).
- [Ope15] OpenStack. *Heat Orchestration Template (HOT) Specification*. OpenStack Orchestration. 2015 (Zitiert auf S. 33).
- [Opp03] D. Oppenheimer. „The importance of understanding distributed system configuration“. In: *Proceedings of the 2003 Conference on Human Factors in Computer Systems Workshop (CHI 2003)*. ACM, Apr. 2003 (Zitiert auf S. 28, 82).
- [Ops16] Opscode, Inc. *Chef Official Site*. Feb. 2016. URL: <http://www.opscode.com/chef> (Zitiert auf S. 12, 52).
- [Par94] D. L. Parnas. „Software Aging“. In: *Proceedings of the 16th International Conference on Software Engineering (ICSE 1994)*. IEEE, Mai 1994, S. 279–287 (Zitiert auf S. 78).
- [PD06] S. Penugonda und R. K. Dash. *SCA application development, Part 2: SCA client and implementation model for Java*. Aug. 2006. URL: <http://www.ibm.com/developerworks/webservices/library/ws-soa-scadev2/> (Zitiert auf S. 31).
- [Pes08] M. Pesic. „Constraint-based Workflow Management Systems: Shifting Control to Users“. Dissertation. Eindhoven University of Technology, Mai 2008 (Zitiert auf S. 51).

- [Pet14] D. Petcu. „Consuming Resources and Services from Multiple Clouds“. In: *Journal of Grid Computing* 12.2 (Jan. 2014), S. 321–345 (Zitiert auf S. 42, 50, 66).
- [PMM15] A. Papaioannou, D. Metallidis und K. Magoutis. „Cross-layer Management of Distributed Applications on Multi-Clouds“. In: *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM 2015)*. IEEE, Mai 2015, S. 552–558 (Zitiert auf S. 55).
- [Pup16] Puppet Labs. *Puppet Official Site*. Feb. 2016. URL: <http://puppetlabs.com/puppet/what-is-puppet> (Zitiert auf S. 12, 52, 68).
- [PZF13] C. Pahl, L. Zhang und F. Fowley. „Interoperability Standards for Cloud Architecture“. In: *Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013)*. SciTePress, Mai 2013, S. 123–126 (Zitiert auf S. 50).
- [RAR13] L. Richardson, M. Amundsen und S. Ruby. *RESTful Web APIs*. O'Reilly, Okt. 2013 (Zitiert auf S. 50).
- [Rei12] R. Reiners. „A Pattern Evolution Process - From Ideas to Patterns“. In: *Informatiktage 2012*. Bd. S-11. LNI. GI, März 2012, S. 115–118 (Zitiert auf S. 91).
- [RHEP12] R. Reiners, R. Halvorsrud, A. W. Eide und D. Pohl. „An Approach to Evolutionary Design Pattern Engineering“. In: *Proceedings of the 19th Conference on Pattern Languages of Programs (PLoP 2012)*. ACM, Okt. 2012 (Zitiert auf S. 91).
- [RJB10] J. Rumbaugh, I. Jacobson und G. Booch. *Unified Modeling Language Reference Manual*. Pearson Education, Apr. 2010 (Zitiert auf S. 34, 36).
- [RSM+07] R. Rebouças, J. Sauvé, A. Moura, C. Bartolini und D. Trastour. „A decision support tool to optimize scheduling of IT changes“. In: *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*. IEEE, Mai 2007, S. 343–352 (Zitiert auf S. 85).
- [SAB+12] S. Strauch, V. Andrikopoulos, U. Breitenbücher, O. Kopp und F. Leymann. „Non-Functional Data Layer Patterns for Cloud Applications“. In: *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2012)*. IEEE, Dez. 2012, S. 601–605 (Zitiert auf S. 91).

- [SAB+13] S. Strauch, V. Andrikopoulos, U. Breitenbücher, S. G. Sáez, O. Kopp und F. Leymann. „Using Patterns to Move the Application Data Layer to the Cloud“. In: *Proceedings of the 5th International Conference on Pervasive Patterns and Applications (PATTERNS 2013)*. Xpert Publishing Services, Mai 2013, S. 26–33 (Zitiert auf S. 91).
- [SABL13] S. Strauch, V. Andrikopoulos, T. Bachmann und F. Leymann. „Migrating Application Data to the Cloud Using Cloud Data Patterns“. In: *Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013)*. SciTePress, Mai 2013, S. 36–46 (Zitiert auf S. 91).
- [Sal05] F. A. Salustri. „Using pattern languages in design engineering“. In: *Proceedings of the 15th International Conference on Engineering Design (ICED 2005)* (Aug. 2005) (Zitiert auf S. 91).
- [Sal16] SaltStack Inc. *Salt Official Site*. Feb. 2016. URL: <http://saltstack.com> (Zitiert auf S. 52).
- [SB98] R. Sturm und W. Bumpus. *Foundations of Application Management*. Wiley, Nov. 1998 (Zitiert auf S. 39).
- [SBK+12] S. Strauch, U. Breitenbücher, O. Kopp, F. Leymann und T. Unger. „Cloud Data Patterns for Confidentiality“. In: *Proceedings of the 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012)*. SciTePress, Apr. 2012, S. 387–394 (Zitiert auf S. 91).
- [Sch10] T. Scheibler. „Ausführbare Integrationsmuster“. Dissertation. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2010 (Zitiert auf S. 96).
- [Sch14] D. Schleicher. „Modellierung regelkonformer Geschäftsprozesse“. Dissertation. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2014 (Zitiert auf S. 116).
- [Sch15] D. Schumm. „Sichten auf Geschäftsprozesse mit besonderer Betrachtung von Compliance“. Dissertation. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2015 (Zitiert auf S. 91).

- [SKK+11] D. Schumm, D. Karastoyanova, O. Kopp, F. Leymann, M. Sonntag und S. Strauch. „Process Fragment Libraries for Easier and Faster Development of Process-based Applications“. In: *Journal of Systems Integration* 2.1 (2011), S. 39–55 (Zitiert auf S. 237).
- [SKLS10] D. Schumm, D. Karastoyanova, F. Leymann und S. Strauch. „Fragmento: Advanced Process Fragment Library“. In: *Proceedings of the 19th International Conference on Information Systems Development (ISD 2010)*. Springer, Sep. 2010, S. 659–670 (Zitiert auf S. 237).
- [SLS10] D. Schumm, F. Leymann und A. Streule. „Process Viewing Patterns“. In: *Proceedings of the 14th International Conference on Enterprise Distributed Object Computing (EDOC 2010)*. IEEE, Okt. 2010, S. 89–98 (Zitiert auf S. 91).
- [SM11] A. Sampaio und N. Mendonça. „Uni4Cloud: An Approach Based on Open Standards for Deployment and Management of Multi-cloud Applications“. In: *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing (SECLOUD 2011)*. ACM, Mai 2011, S. 15–21 (Zitiert auf S. 33, 50).
- [SP08] K. Scheibenberger und I. Pansa. „Modelling Dependencies of IT Infrastructure Elements“. In: *Proceedings of the 3rd IEEE/IFIP International Workshop on Business-driven IT Management (BDIM 2008)*. IEEE, Apr. 2008, S. 112–113 (Zitiert auf S. 285).
- [SRC14] G. C. Silva, L. M. Rose und R. Calinescu. „Cloud DSL: A Language for Supporting Cloud Portability by Describing Cloud Entities“. In: *Proceedings of the 2nd International Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE 2014)*. CEUR-WS.org, Sep. 2014, S. 36–45 (Zitiert auf S. 36).
- [SSSL12] M. Smit, M. Shtern, B. Simmons und M. Litoiu. „Partitioning Applications for Hybrid and Federated Clouds“. In: *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2012)*. IBM, Nov. 2012, S. 27–41 (Zitiert auf S. 42).
- [STI+06] C. S. Shankar, V. Talwar, S. Iyer, Y. Chen, D. S. Milojevic und R. H. Campbell. „Specification-Enhanced Policies for Automated Management of Changes in IT Systems“. In: *Proceedings of the 20th Conference on*

Systems Administration (LISA 2006). USENIX, Dez. 2006, S. 103–118 (Zitiert auf S. 42).

- [Tab05] P. Tabeling. *Softwaresysteme und ihre Modellierung: Grundlagen, Methoden und Techniken*. Springer, Juli 2005 (Zitiert auf S. 26).
- [TFL09] D. Trastour, R. Fink und F. Liu. „ChangeRefinery: Assisted Refinement of High-Level IT Change Requests“. In: *Proceedings of the IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY 2009)*. IEEE, Juli 2009, S. 68–75 (Zitiert auf S. 80, 97).
- [TRB07] D. Trastour, M. Rahmouni und C. Bartolini. „Activity-Based Scheduling of IT Changes“. In: *Inter-Domain Management*. Bd. 4543. Springer, 2007, S. 73–84 (Zitiert auf S. 85).
- [TRH+04] D. Trowbridge, U. Roxburgh, G. Hohpe, D. Manolescu und E. Nadhan. *Integration Patterns*. Patterns & Practices. Prentice Hall, 2004 (Zitiert auf S. 91).
- [Tur14] J. Turnbull. *The Docker Book*. James Turnbull, 2014 (Zitiert auf S. 100).
- [TV14] M. Taylor und S. Vargo. *Learning Chef: A Guide to Configuration Management and Automation*. O'Reilly, Nov. 2014 (Zitiert auf S. 52).
- [Uph14] T. Uphill. *Mastering Puppet*. Packt Publishing, Juli 2014 (Zitiert auf S. 52).
- [vABtHK00] W. M. P. van der Aalst, A. P. Barros, A. H. M. ter Hofstede und B. Kiepuszewski. „Advanced Workflow Patterns“. In: *Proceedings of the 7th International Conference on Cooperative Information Systems (CoopIS 2000)*. Springer, Sep. 2000, S. 18–29 (Zitiert auf S. 91).
- [vAtH05] W. M. P. van der Aalst und A. H. M. ter Hofstede. „YAWL: Yet Another Workflow Language“. In: *Information Systems* 30.4 (Juni 2005), S. 245–275 (Zitiert auf S. 71).
- [vAtHKB03] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski und A. P. Barros. „Workflow Patterns“. In: *Distributed and Parallel Databases* 14 (Juli 2003), S. 5–51 (Zitiert auf S. 91, 225).
- [vHee14] U. van Heesch. *Open Pattern Repository*. Mai 2014. URL: <https://code.google.com/p/openpatternrepository/> (Zitiert auf S. 91).

- [vJBN+10] H. von Jouanne-Diedrich, J. Blechinger, C. P. Neumann, S. Schwarz und R. Lenz. „Integration verteilter und heterogener Configuration-Management-Datenbanken“. In: *Informatik-Spektrum* 33.4 (Aug. 2010), S. 351–362 (Zitiert auf S. 29).
- [VRB11] L. M. Vaquero, L. Roderio-Merino und R. Buyya. „Dynamically Scaling Applications in the Cloud“. In: *SIGCOMM Computer Communication Review* 41.1 (Jan. 2011), S. 45–52 (Zitiert auf S. 44).
- [VVK08] J. Vanhatalo, H. Völzer und J. Koehler. „The Refined Process Structure Tree“. In: *Business Process Management*. Bd. 5240. Springer, 2008, S. 100–115 (Zitiert auf S. 237).
- [VVL07] J. Vanhatalo, H. Völzer und F. Leymann. „Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition“. In: *Proceedings of the Fifth International Conference on Service-Oriented Computing (ICSOC 2007)*. Springer, Sep. 2007, S. 43–55 (Zitiert auf S. 237).
- [W3C99] W3C. *XSL Transformations (XSLT) Version 1.0*. World Wide Web Consortium (W3C). 1999 (Zitiert auf S. 252).
- [WBB+14a] J. Wettinger, T. Binz, U. Breitenbücher, O. Kopp und F. Leymann. „Streamlining Cloud Management Automation by Unifying the Invocation of Scripts and Services Based on TOSCA“. In: *International Journal of Organizational and Collective Intelligence (IJOICI)* 4.2 (Apr. 2014), S. 45–63 (Zitiert auf S. 13, 72, 230, 292, 299).
- [WBB+14b] J. Wettinger, T. Binz, U. Breitenbücher, O. Kopp, F. Leymann und M. Zimmermann. „Unified Invocation of Scripts and Services for Provisioning, Deployment, and Management of Cloud Applications Based on TOSCA“. In: *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014)*. SciTePress, Apr. 2014, S. 559–568 (Zitiert auf S. 72, 230, 292, 299).
- [WBB15] D. Weerasiri, B. Benatallah und M. Barukh. „Process-driven Configuration of Federated Cloud Resources“. In: *Database Systems for Advanced Applications*. Bd. 9049. Springer, 2015, S. 334–350 (Zitiert auf S. 42, 72).

- [WBL+09] J. A. Wickboldt, L. A. Bianchin, R. C. Lunardi, F. G. Andreis, W. L. da Costa Cordeiro, C. B. Both, L. Z. Granville, L. P. Gaspary, D. Trastour und C. Bartolini. „Improving IT Change Management Processes with Automated Risk Assessment“. In: *Integrated Management of Systems, Services, Processes and People in IT*. Bd. 5841. Springer, 2009, S. 71–84 (Zitiert auf S. [40](#), [82](#), [83](#)).
- [WBL14a] J. Wettinger, U. Breitenbücher und F. Leymann. „Compensation-based vs. Convergent Deployment Automation for Services Operated in the Cloud“. In: *Proceedings of the 12th International Conference on Service-Oriented Computing (ICSOC 2014)*. Springer, Nov. 2014, S. 336–350 (Zitiert auf S. [68](#), [69](#)).
- [WBL14b] J. Wettinger, U. Breitenbücher und F. Leymann. „Standards-based Dev-Ops Automation and Integration Using TOSCA“. In: *Proceedings of the 7th International Conference on Utility and Cloud Computing (UCC 2014)*. IEEE, Dez. 2014, S. 59–68 (Zitiert auf S. [231](#)).
- [WBL15a] J. Wettinger, U. Breitenbücher und F. Leymann. „Any2API - Automated APIfication“. In: *Proceedings of the 5th International Conference on Cloud Computing and Services Science (CLOSER 2015)*. SciTePress, Mai 2015, S. 475–486 (Zitiert auf S. [231](#), [292](#), [299](#)).
- [WBL15b] J. Wettinger, U. Breitenbücher und F. Leymann. „Compensation and Convergence - Comparing and Combining Deployment Automation Approaches“. In: *International Journal of Cooperative Information Systems* 24.03 (Aug. 2015) (Zitiert auf S. [69](#)).
- [Wel94] D. S. Weld. „An Introduction to Least Commitment Planning“. In: *AI Magazine* 15.4 (1994), S. 27–61 (Zitiert auf S. [56](#), [57](#), [214](#), [218](#), [234](#)).
- [WF12] T. Wellhausen und A. Fiesser. „How to Write a Pattern?: A Rough Guide for First-time Pattern Authors“. In: *Proceedings of the 16th European Conference on Pattern Languages of Programs (EuroPLOP 2011)*. ACM, Juli 2012 (Zitiert auf S. [91](#)).
- [Wil12] B. Wilder. *Cloud Architecture Patterns*. O'Reilly Media, Sep. 2012 (Zitiert auf S. [91](#)).

- [WKZ14] S. Wang, I. Keivanloo und Y. Zou. „How Do Developers React to RESTful API Evolution?“ In: *Proceedings of the 12th International Conference on Service-Oriented Computing (ICSOC 2014)*. Springer, Nov. 2014, S. 245–259 (Zitiert auf S. 50).
- [WMdC+09] J. A. Wickboldt, G. S. Machado, W. L. da Costa Cordeiro, R. C. Lunardi, A. D. dos Santos, F. G. Andreis, C. B. Both, L. Z. Granville, L. P. Gaspary, C. Bartolini und D. Trastour. „A solution to support risk analysis on IT Change Management“. In: *Proceedings of the 11th IFIP/IEEE International Symposium on Integrated Network Management (IM 2009)*. IEEE, Juni 2009, S. 445–452 (Zitiert auf S. 83).
- [WWB+13] T. Waizenegger, M. Wieland, T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, B. Mitschang, A. Nowak und S. Wagner. „Policy4TOSCA: A Policy-Aware Cloud Service Provisioning Approach to Enable Secure Cloud Computing“. In: *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*. Springer, Sep. 2013, S. 360–376 (Zitiert auf S. 44).
- [WWF+12] I. Weber, H. Wada, A. Fekete, A. Liu und L. Bass. „Automatic Undo for Cloud Management via AI Planning“. In: *Proceedings of the Eighth Workshop on Hot Topics in System Dependability (HotDep 2012)*. USENIX, Okt. 2012 (Zitiert auf S. 86, 233).
- [WWF+13] I. Weber, H. Wada, A. Fekete, A. Liu und L. Bass. „Supporting Undoability in Systems Operations“. In: *Proceedings of the 27th International Conference on Large Installation System Administration (LISA 2013)*. USENIX, 2013, S. 75–87 (Zitiert auf S. 87, 233).
- [XWB+13] X. Xu, I. Weber, L. Bass, L. Zhu, H. Wada und F. Teng. „Detecting Cloud Provisioning Errors Using an Annotated Process Model“. In: *Proceedings of the 8th Workshop on Middleware for Next Generation Internet Computing (MW4NextGen 2013)*. ACM, Dez. 2013, 5:1–5:6 (Zitiert auf S. 82).
- [XZL+13] X. Xu, L. Zhu, J. Li, L. Bass, Q. Lu und M. Fu. „Modeling and Analysing Operation Processes for Dependability“. In: *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2013)*. IEEE, Juni 2013, S. 1–2 (Zitiert auf S. 82).

- [ZA05] U. Zdun und P. Avgeriou. „Modeling Architectural Patterns Using Architectural Primitives“. In: *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA 2005)*. ACM, Okt. 2005, S. 133–146 (Zitiert auf S. [259](#), [260](#)).
- [ZAHD08] U. Zdun, P. Avgeriou, C. Hentrich und S. Dustdar. „Architecting as Decision Making with Patterns and Primitives“. In: *Proceedings of the 3rd International Workshop on Sharing and Reusing Architectural Knowledge (SHARK 2008)*. ACM, Mai 2008, S. 11–18 (Zitiert auf S. [259](#)).
- [Zdu07] U. Zdun. „Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis“. In: *Software: Practice & Experience* 37.9 (Juli 2007), S. 983–1016 (Zitiert auf S. [91](#)).

Alle URLs wurden zuletzt am 01.02.2016 geprüft.

ABBILDUNGSVERZEICHNIS

1.1	Vision der musterbasierten Managementautomatisierung	14
1.2	Überblick der Forschungsbeiträge	16
2.1	Entwicklungs- und Betriebsphase des Anwendungslebenszyklus	38
2.2	Kopplung von Managementplänen an Schnittstellen.	76
3.1	Schritte und involvierte Artefakte der PALMA-Methode	103
3.2	Variante der PALMA-Methode	112
3.3	Umsetzung und Automatisierung der PALMA-Methode	115
4.1	Konzept der DMMN-Modellierung	120
4.2	Schichten, Klassen und Relationen des DMMN-Metamodells . . .	123
4.3	Darstellung eines Komponentenelements	156
4.4	Alternative Darstellungsformen für Komponentenelemente . . .	157
4.5	Darstellung eines Relationenelements	158
4.6	Darstellung von Annotationenelementen	159
4.7	Beispiel eines DMMN-Modells	161
5.1	Grundkonzept der Plangenerierung	166
5.2	Architektur des Management-Planlet Frameworks	167

5.3	Konzeptioneller Aufbau eines Management-Planlets	169
5.4	Beispiel eines Management-Planlets	170
5.5	Verhältnis von Plänen, Planlets und Managementoperationen . .	172
5.6	Beispiel eines generischen Management-Planlets zur Bereitstel- lung von Komponenten	178
5.7	Beispiel eines generischen Management-Planlets zur Herstellung einer SQL-Verbindung	179
5.8	Transformationsmethode	182
5.9	Evaluation der Kompatibilität zwischen DMMN-Modellen	191
5.10	Illustration des Algorithmus zur Transformation deklarativer DMMN-Modelle in imperative Managementmodelle	195
5.11	Illustration des Algorithmus zur Planlet-Konvertierung	209
5.12	Generierung eines imperativen Plans aus $\emptyset$ und $\mathcal{A}$	227
5.13	Technische Umsetzung von Plänen und Planlets in BPEL	228
6.1	Grundkonzept der Automatisierung von Managementmustern .	240
6.2	Erweiterung des PALMA-Frameworks um Musterunterstützung .	241
6.3	Aufbau eines automatisierten Managementmusters	243
6.4	Beispiel eines automatisierten Managementmusters	253
6.5	Eingabe und Resultat der Transformation des Beispielmusters .	255
6.6	Beispiel des verfeinerten Managementmusters	257
6.7	Eingabe und Resultat der Transformation des verfeinerten auto- matisierten Managementmusters	258
6.8	Generiertes DMMN-Modell des Bereitstellungsmusters	261
6.9	DMMN-Modell des technologiespezifischen Musters	263
6.10	Planlets zur Ausführung der Annotationen des technologiespe- zifischen automatisierten Bereitstellungsmusters	265
7.1	Grundkonzept der automatisierten Analysemuster	274
7.2	Erweiterung des Frameworks um automatisierte Analysemuster	275
7.3	Aufbau eines automatisierten Analyseusters	278
7.4	Beispiel eines automatisierten Analyseusters	283

8.1	Architektur des PALMA-Frameworks	288
8.2	Grafisches Werkzeug zur Modellierung von DMMN-Modellen . . .	289
8.3	PALMA-Framework kombiniert mit Integrationsstechnologien . .	292

SYMBOLVERZEICHNIS

Notation	Bedeutung
K_d	Menge aller Komponentenelemente eines DMMN-Modells d , siehe Definition 15 .
R_d	Menge aller Relationenelemente eines DMMN-Modells d , siehe Definition 16 .
TE_d	Menge aller Topologieelemente eines DMMN-Modells d , d. h. Vereinigungsmenge aller Komponenten- und Relationenelemente in d , siehe Definition 17 .
TE	Menge aller Topologieelemente aller DMMN-Modelle, siehe Definition 18 .
KT_d	Menge aller Komponententypenelemente eines DMMN-Modells d , siehe Definition 19 .
RT_d	Menge aller Relationentypenelemente eines DMMN-Modells d , siehe Definition 20 .
TET_d	Vereinigungsmenge aller Komponenten- und Relationentypenelemente eines DMMN-Modells d , siehe Definition 21 .
MS_d	Vereinigungsmenge aller Topologie- und Typelemente eines DMMN-Modells d , siehe Definition 25 .
E_d	Menge aller Eigenschaftenelemente eines DMMN-Modells d , siehe Definition 26 .
E	Menge aller Eigenschaftenelemente aller DMMN-Modelle, siehe Definition 28 .

Notation	Bedeutung
I_d	Menge aller Managementschnittstellenelemente eines DMMN-Modells d , siehe Definition 32 .
O_d	Menge aller Operationenelemente eines DMMN-Modells d , siehe Definition 35 .
A_d	Menge aller Artefaktelemente eines DMMN-Modells d , siehe Definition 39 .
MA_d	Menge aller Managementannotationenelemente eines DMMN-Modells d , siehe Definition 43 .
C_d	Menge aller Konfigurationenelemente eines DMMN-Modells d , siehe Definition 46 .
P_d	Menge aller Reihenfolgespezifikationenelemente eines DMMN-Modells d , siehe Definition 49 .
P	Menge aller Reihenfolgespezifikationenelemente aller DMMN-Modelle, siehe Definition 50 .
typ_d	Abbildung, die jedem Topologieelement in einem DMMN-Modell d ihr Typelement zuordnet, siehe Definition 22 .
$supertyp_d$	Abbildung, die jedem Komponententyp- bzw. Relationentypelement in einem DMMN-Modell d sein Supertypeelement zuordnet, siehe Definition 23 .
$supertypen$	Abbildung, die jedes Topologieelement $te_i \in TE_d$ in einem DMMN-Modell d auf alle Typelemente $tet_j \in TET_d$ abbildet, von denen es direkt oder transitiv erbt, siehe Definition 24 .
$eigeneEigenschaften_d$	Abbildung, die jedem Komponenten-, Relationen-, Komponententyp- oder Relationentypelement in einem DMMN-Modell d seine Eigenschaftenelemente zuordnet, siehe Definition 27 .
$eigenschaften$	Abbildung, die jedes Topologieelement unter Einhaltung der Vererbungsgesetze auf dessen Eigenschaftenelemente abbildet, siehe Definition 31 .

Notation	Bedeutung
$eigeneSchnittstellen_d$	Abbildung, die jedem Topologieelement in einem DMMN-Modell d seine Managementschnittstellenelemente zuordnet, siehe Definition 33 .
$schnittstellen$	Abbildung, die jedes Topologieelement unter Einhaltung der Vererbungsgesetze auf dessen Schnittstellenelemente abbildet, siehe Definition 38 .
$operationen_d$	Abbildung, die jedem Managementschnittstellenelement in einem DMMN-Modell d seine Operationenelemente zuordnet, siehe Definition 36 .
$Parameter$	Menge aller Parameter von Managementoperationenelementen, siehe Definition 35 .
$eigeneArtefakte_d$	Abbildung, die jedem Topologie- oder Operationenelement in einem DMMN-Modell d dessen Artefaktelemente zuordnet, siehe Definition 40 .
$artefakte$	Abbildung, die jedes Topologieelement unter Einhaltung der Vererbungsgesetze auf dessen Artefaktelemente abbildet, siehe Definition 42 .
$annotationen_d$	Abbildung, die jedem Topologieelement in einem DMMN-Modell d seine Annotationenelemente zuordnet, siehe Definition 44 .
$\uplus_i$	Vererbungsoperator, siehe Definition 29 .
MP	Menge aller Management-Planlets, siehe Definition 53 .
EF_{mp}	Menge aller Effektelemente eines Management-Planlets mp , siehe Definition 53 .
EF	Menge aller Effektelemente aller Management-Planlets, siehe Definition 53 .
tf_{mp}	Topologiefragment eines Management-Planlets mp , siehe Definition 53 .
$effekte_{mp}$	Abbildung, die jedem Topologieelement in einem Topologiefragment eines Management-Planlets mp die zugehörigen Effektelemente zuweist, siehe Definition 53 .

Notation	Bedeutung
$\tilde{\approx}_a \subseteq A \times A$	Kompatibilitätsrelation zum Vergleich von Artefaktelementen, siehe Definition 58 .
$\tilde{\approx}_o \subseteq O \times O$	Kompatibilitätsrelation zum Vergleich von Operationenelementen, siehe Definition 59 .
$\tilde{\approx}_i \subseteq I \times I$	Kompatibilitätsrelation zum Vergleich von Schnittstellenelementen, siehe Definition 60 .
$\tilde{\approx}_{ma} \subseteq MA \times MA$	Kompatibilitätsrelation zum Vergleich von Managementannotationenelementen, siehe Definition 61 .
$\tilde{\approx}_{te} \subseteq TE \times TE$	Kompatibilitätsrelation zum Vergleich von Topologieelementen, siehe Definition 62 .
$\tilde{\approx}_{ef} \subseteq EF \times E$	Kompatibilitätsrelation zur Überprüfung, ob ein Effektelement $ef \in EF$ eines Planlets kompatibel zu einer Vorbedingung eines anderen Planlets in Form eines Eigenschaftenelements $e \in E$ ist, siehe Definition 70 .
$\tilde{\approx}_e \subseteq E \times E$	Kompatibilitätsrelation zum Vergleich von Eigenschaftenelementen, siehe Definition 73 .
AMM	Menge aller automatisierten Managementmuster, siehe Definition 72 .
AAM	Menge aller automatisierten Analysemuster, siehe Definition 80 .
AA	Menge aller Analysealgorithmen, siehe Definition 80 .
$\tilde{\approx}_M \subseteq AMM \times D$	Kompatibilitätsrelation zur Überprüfung, ob ein automatisiertes Managementmuster auf ein DMMN-Modell angewendet werden kann, siehe Definition 77 .
$\tilde{\approx}_{AM} \subseteq AAM \times D$	Kompatibilitätsrelation zur Überprüfung, ob ein automatisiertes Analysemuster auf ein DMMN-Modell angewendet werden kann, siehe Definition 81 .
$\tilde{\approx}_\circ \subseteq D \times D$	Kompatibilitätsrelation zum statischen Vergleich von DMMN-Modellen, siehe Definition 67 .
$\tilde{\approx}_\bullet \subseteq D \times D$	Kompatibilitätsrelation zum dynamischen Vergleich von DMMN-Modellen, siehe Definition 75 .

Notation	Bedeutung
<i>kompatibel</i> [*]	Relation zur Feststellung der dynamischen Kompatibilität zweier DMMN-Modelle bezüglich einer statisch kompatiblen Zuordnung, siehe Definition 74 .
$\mathfrak{M}^s := \wp(TE \times TE)$	Menge aller strukturellen Zuordnungen zwischen Topologieelementen, siehe Definition 63 .
$\mathfrak{M}^{ab} := \wp(MA \times MA)$	Menge aller aufgabenbezogenen Zuordnungen zwischen Annotationenelementen, siehe Definition 65 .
$\mathfrak{M}_{d_1, d_2}^{\approx_\circ} \subseteq \mathfrak{M}^s \times \mathfrak{M}^{ab}$	Menge aller statisch kompatiblen Zuordnungen zwischen einem DMMN-Modell d_1 und einem DMMN-Modell d_2 , siehe Definition 68 .
$\mathfrak{M}_{d_1, d_2}^{\approx_\bullet} \subseteq \mathfrak{M}^s \times \mathfrak{M}^{ab}$	Menge aller dynamisch kompatiblen Zuordnungen zwischen einem DMMN-Modell d_1 und einem DMMN-Modell d_2 , siehe Definition 76 .
$\mathfrak{M}_{tf_{aam}, d}^{\approx_\bullet \otimes} \subseteq \mathfrak{M}^s \times \mathfrak{M}^{ab}$	Menge aller problemidentifizierenden Zuordnungen zwischen einem Topologiefragment tf_{aam} eines automatisierten Analyseusters $aam \in AAM$ und einem DMMN-Modell $d \in D$, siehe Definition 83 .
$MP_d^{\approx_\circ}$	Menge aller Management-Planlets, die zu einem DMMN-Modell d statisch kompatibel sind, siehe Definition 69 .
T	Menge aller Transformationen aller automatisierten Muster, siehe Definition 72 .
t_m	Abbildung, welche die Lösung des Musters m als automatisierte Transformation beschreibt, siehe Definition 72 .