Universität Stuttgart

Crawling von Enterprise Topologien zur automatisierten Migration von Anwendungen – eine Cloud-Perspektive

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

Tobias Binz

aus Stuttgart

Hauptberichter: Prof. Dr. Frank Leymann

Mitberichter: Univ. Prof. Dr. Schahram Dustdar

Tag der mündlichen Prüfung: 16. April 2015

Institut für Architektur von Anwendungssystemen
der Universität Stuttgart

2015

Inhaltsverzeichnis

1 Einleitung **15**
1.1 Problemstellungen und Forschungsbeiträge 17
1.1.1 Methode zur Migration von Anwendungen 18
1.1.2 Konzept zur Repräsentation und Verarbeitung von IT-Instanzmodellen 18
1.1.3 Crawling von IT-Instanzmodellen 19
1.1.4 Identifikation und Isolation der zu migrierenden Anwendung 20
1.1.5 Konzept zur Umsetzung und Automatisierung der AROMA-Methode mittels TOSCA 21
1.1.6 Architektur, Realisierung und Validierung der Beiträge . 22
1.2 Aufbau der Arbeit 23
1.3 Veröffentlichungen 24

2 Grundlagen und verwandte Arbeiten **29**
2.1 Service-orientierte Architekturen 30
2.2 Anwendungen 31

2.3 Cloud Computing 33
2.3.1 Cloud-Eigenschaften 34
2.3.2 Cloud-Anwendungen 36
2.4 Anwendungstopologien 38
2.4.1 Komponentenbasierte Softwareentwicklung 39
2.4.2 Automatisierte Bereitstellung von Anwendungen 39
2.4.3 Topology and Orchestration Specification for Cloud Applications 41
2.5 IT-Instanzmodelle 45
2.5.1 Abgrenzung von Anwendungstopologien 46
2.5.2 Abgrenzung von Enterprise Architecture Management . . 46
2.5.3 Verwandte Arbeiten zu IT-Instanzmodellen 47
2.5.4 Diskussion und offene Forschungsfragen 48
2.6 Erstellung von Enterprise Topologien 49
2.6.1 Automatisierte Enterprise Architektur Dokumentation . . 50
2.6.2 Software Architecture Reconstruction 51
2.6.3 Identifikation von Abhängigkeiten zwischen Diensten . . 52
2.6.4 Erstellung von Instanzmodellen großer Teile der IT . . . 54
2.6.5 Komponentenspezifische Extraktion 56
2.6.6 Diskussion und offene Forschungsfragen 57
2.7 Migration von Anwendungen 58
2.7.1 Migrationsstrategien 59
2.7.2 Vorgehensmodelle zur Anwendungsmigration 60
2.7.3 Automatisierung der Migrationsdurchführung 63

3 Methode zur Migration von Anwendungen 67
3.1 Anforderungen und Herausforderungen 68
3.2 AROMA-Methode 69
3.2.1 Schritt 1: Crawling der IT 70
3.2.2 Schritt 2: Identifikation, Partitionierung und Isolation der Anwendung aus der Enterprise Topologie 72
3.2.3 Schritt 3: Transformation in Anwendungstopologie . . . 73
3.2.4 Schritt 4: Adaption an die konkrete Zielumgebung 74

3.2.5 Schritt 5: Evaluation und manuelle Anpassung 76
3.2.6 Schritt 6: Paketierung 77
3.2.7 Schritt 7: Bereitstellung und Umstellung 78
3.3 Varianten der Methode 80
3.3.1 Variante 1: Selektive Erstellung der Enterprise Topologie 80
3.3.2 Variante 2: Migration geschäftsprozessbasierter Anwendungen 81
3.4 Diskussion und Zusammenfassung 84
3.4.1 Für die Migration geeignete Anwendungen 85
3.4.2 Für die Migration geeignete Zielumgebungen 86

4 Enterprise Topologie Graph 89
4.1 ETG-Metamodell 91
4.2 Definition von Enterprise Topologie Graphen 92
4.2.1 Komponenten 94
4.2.2 Relationen 94
4.2.3 Typen 95
4.2.4 Elemente und ihre Eigenschaften 98
4.2.5 Grundlegende Relationentypen 99
4.2.6 Segmente 100
4.3 Granularität von Enterprise Topologie Graphen 101
4.4 Topologie Queries 103
4.5 Operation DeepDive 107
4.6 Identifikation und Isolation von Komponenten aus dem ETG . . 108
4.6.1 Schritt 1: Bestimmung der potentiellen Komponenten . . 109
4.6.2 Schritt 2: Reduktion der Komponenten 109
4.6.3 Schritt 3: Behandlung gemeinsam genutzter Komponenten 112
4.6.4 Erweiterbarkeit 114
4.6.5 Diskussion 114
4.7 Diskussion und Zusammenfassung 115

5 Crawling von Enterprise Topologie Graphen 117
5.1 Crawler-Methode 119
5.1.1 Anforderungen an die Crawler-Methode 119
5.1.2 Konzept 121
5.1.3 Crawler-Architektur 125
5.1.4 Rollen 127
5.1.5 Ablaufsteuerung 127
5.1.6 Deduplizierung und Konsolidierung 136
5.2 Arbeitsweise von Crawler-Plugins 140
5.2.1 BPEL und WSDL 141
5.2.2 Anwendung und Webserver 143
5.2.3 Enterprise Service Bus 145
5.2.4 Betriebssystem und Server 149
5.3 Entwicklung von Crawler-Plugins 151
5.3.1 Entwicklungs- und Testmethode 151
5.3.2 Informationsquellen für Crawler-Plugins 153
5.4 Evaluation und Validierung 156
5.4.1 Validierung der Crawler-Methode 156
5.4.2 Zugriffsrechte und Sicherheit 157
5.4.3 Erweiterbarkeit und Integration 158
5.4.4 Aktualisierung von ETGs 158
5.4.5 Einfluss auf Produktionssysteme minimieren 159
5.4.6 ETG-Qualität 160
5.5 Diskussion und Zusammenfassung 161

6 Umsetzung der AROMA-Methode mit TOSCA 163
6.1 Typsystem für ETG und TOSCA 164
6.2 TOSCA-Ökosystem „OpenTOSCA“ 165
6.3 Umsetzung der Schritte der AROMA-Methode mit TOSCA . . . 166
6.3.1 Schritt 3: Transformation in Anwendungstopologie . . . 166
6.3.2 Schritt 4: Adaption an die konkrete Zielumgebung 168
6.3.3 Schritt 5: Evaluation und manuelle Anpassung 173
6.3.4 Schritt 6: Paketierung 173

6.3.5 Schritt 7: Bereitstellung und Umstellung 173

7 Architektur und Prototypen **175**
7.1 Gesamtarchitektur und ETG-Framework 175
7.2 ETG-Verwaltung 178
7.3 ETG-Crawler 181
7.3.1 Architektur des ETG-Crawlers 181
7.3.2 Realisierung des ETG-Crawlers 182
7.3.3 Entwickelte Crawler-Plugins 183
7.4 AROMA-Migrationsassistent 186
7.4.1 Architektur des Migrationsassistenten 187
7.4.2 Realisierung des Migrationsassistenten 188
7.4.3 Realisierung von Variante 2 der AROMA-Methode 191

8 Validierung und Evaluation **193**
8.1 Fallstudie: Migration der Anwendung „Moodle“ 194
8.2 Verwendung von Enterprise Topologie Graphen und deren Crawling in anderen Arbeiten 198
8.2.1 Deklarative Verwaltung und Adaption laufender Cloud-Anwendungen 198
8.2.2 Reengineering von Geschäftsprozessen anhand ökologischer Faktoren 198
8.3 Evaluation und Schlussfolgerungen 199
8.3.1 Automatisierung 199
8.3.2 Korrektheit 200
8.3.3 Verbesserung der Nutzung der Cloud-Eigenschaften . . . 201
8.3.4 Verbesserung der Portabilität 202
8.3.5 Anwendbarkeit 203
8.3.6 Erweiterbarkeit 205

9 Zusammenfassung und Ausblick **207**
9.1 Zusammenfassung der Forschungsbeiträge 208
9.2 Ausblick 210

Literaturverzeichnis **213**

Abbildungsverzeichnis **243**

Definitionsverzeichnis **245**

Algorithmenverzeichnis **247**

Liste mathematischer Symbole **249**

Zusammenfassung

Eine schnelle Anpassung der IT an sich ändernde Anforderungen bei gleichzeitiger Reduktion der Kosten bestimmt heute die Konkurrenzfähigkeit einer Organisation. Voraussetzung dafür ist ein technisch detaillierter Einblick in die gesamte IT, also ein Instanzmodell aller Komponenten und deren Beziehungen zueinander. Da Organisationen diese Art der Dokumentation meist nicht durchführen, sind diese IT-Instanzmodelle typischerweise nicht vorhanden, unvollständig oder veraltet. Eine Ursache dafür ist, dass die manuelle Identifikation von Komponenten und deren Beziehungen eine sehr zeitaufwändige, fehleranfällige und somit kostenintensive Aufgabe ist. Neben der Adaption der IT im Allgemeinen erschwert dies auch die Migration von Anwendungen, welche durch den Trend zum Auslagern der IT in die Cloud stark nachgefragt wird. Die Vision dieser Arbeit ist es, einen technisch detaillierten, vollständigen und aktuellen *Einblick* in die IT zu erlauben und diesen zu nutzen, um die automatisierte *Migration von Anwendungen* zu ermöglichen.

Dafür stellt die vorliegende Arbeit eine Methode zum automatisierten Crawling eines Instanzmodells der gesamten IT einer Organisation vor. Zu dessen Repräsentation, Verwaltung und Verarbeitung wird mit dem Enterprise Topologie Graph (ETG) ein Metamodell eingeführt, das alle Anwendungen, der für deren Betrieb nötigen Komponenten und deren Beziehungen

untereinander repräsentiert. ETGs und ihr automatisiertes Crawling erlauben einen umfassenden und vollständigen Einblick in die IT einer Organisation und bilden somit eine solide Grundlage für deren Analyse, Adaption und Optimierung. Darauf aufbauend wird eine Methode zur Migration von Anwendungen (AROMA) entwickelt, die es ermöglicht, von den Vorteilen fortschrittlicher IT-Umgebungen zu profitieren, ohne diese Anwendungen neu entwickeln zu müssen. Nach dem Crawling des ETGs der Ursprungsumgebung wird in der AROMA-Methode die zu migrierende Anwendung extrahiert, transformiert, evaluiert, adaptiert und in der Zielumgebung, zum Beispiel einer Cloud, bereitgestellt. Die Umsetzung der AROMA-Methode mithilfe des OASIS-Standards TOSCA trägt zur Automatisierung der Migration bei und erhält die Funktionalität der Anwendung. Die Forschungsbeiträge und Prototypen werden durch verschiedene Fallstudien validiert und anhand der Aspekte Automatisierung, Korrektheit, Anwendbarkeit, Erweiterbarkeit sowie der Verbesserung der Cloud-Eigenschaften und Portabilität der Anwendung evaluiert.

Abstract

In order to remain competetive today, organizations have to ensure a fast adaptation of IT to changing demand while at the same time reducing IT cost. However, this requires a technically detailed overview of the complete IT, i.e., an instance model of all IT components and their relations. Typically, this kind of documentation is not performed in organizations and, therefore, these IT instance models are incomplete, outdated, or do not exist at all. One reason for this is that the manual identification of components and their relations is time consuming, error-prone, and hence costly. Besides IT adaptation in general, the lack of insight into IT is an obstacle for the migration of applications. Due to the trend of outsourcing IT into the cloud, migrating applications is on high demand. The vision of this thesis is to enable a technically-detailed, complete and up-to-date *insight* into an organization's IT and, using this insight, to enable *application migration*.

Therefore, this thesis introduces a method for the automated crawling of instance models, representing the whole IT of an organization. To represent, manage, and process these instance models, Enterprise Topology Graphs (ETG) are proposed, a meta model to represent all applications, the components required to operate them, and their relations to each other. ETGs and their automated crawling enable a comprehensive and complete insight into an organization's IT and thus, provide a sound base to analyze, adapt,

and optimize IT in general. On top of this, a method for the migration of applications (AROMA) is developed, which enables organizations to benefit from advanced IT environments, without the need to reimplement their existing applications. After crawling the ETG from the source environment, an application migrated using the AROMA method is extracted, transformed, evaluated, adapted, and deployed in its target environment, e.g., a cloud. The realization of the AROMA method based on the OASIS standard TOSCA helps to automate all steps of the migration and preserves the functionality of the application. The research contributions and prototypical implementation are validated by various case studies and evaluated in terms of automation, correctness, general applicability, extensibility as well as the application's improved exploitation of cloud-properties and portability.

Danksagungen

Ohne die vielfältige Unterstützung einer Reihe von Personen wäre diese Arbeit nicht möglich gewesen. Ich möchte zuallererst Prof. Dr. Frank Leymann für die Möglichkeit danken, am IAAS arbeiten und forschen zu können, sowie für das mir entgegengebrachte Vertrauen, den überlassenen Freiraum und die übertragene Verantwortung. Danke für die zahlreichen intensiven Diskussionen, deine Begeisterungsfähigkeit und Motivation. Mein Dank gilt außerdem Univ. Prof. Dr. Schahram Dustdar für die Übernahme der Zweitkorrektur meiner Arbeit. Meinen Kollegen am IAAS danke ich für die gute Zusammenarbeit, insbesondere Uwe Breitenbücher, Oliver Kopp und David Schumm für die vielfältige Unterstützung und die angeregten Diskussionen. Außerdem möchte ich allen danken, die mir beim Korrekturlesen der Arbeit zur Seite gestanden haben. Last but not least danke ich meiner Familie und meinen Freunden für ihre nicht nachlassende Ermutigung und Geduld, die für das gesamte Promotionsvorhaben und Schreiben der Dissertation entscheidend waren.

KAPITEL 1

EINLEITUNG

Informationstechnologie (IT) spielt mittlerweile in den meisten Bereichen einer Organisation eine zentrale Rolle und gewinnt stetig an Bedeutung. In gleichem Maße steigen jedoch auch die Kosten für den Betrieb und die Weiterentwicklung der IT. Die Einführung neuer Produkte und Technologien, Änderungen an Geschäftsabläufen sowie Aufkäufe und Fusionen lassen die Komplexität und Kosten weiter steigen [GMR10]. Die Wettbewerbsfähigkeit einer Organisation hängt entscheidend davon ab, wie schnell die IT adaptiert und an neue Gegebenheiten angepasst werden kann. Grundlage dafür ist ein detaillierter Einblick in die technische Struktur und die Abhängigkeiten der gesamten IT, welcher durch den ersten Forschungsschwerpunkt der vorliegenden Arbeit adressiert wird. Dieser führt mit dem Enterprise Topologie Graph (ETG) ein Metamodell zur Repräsentation und Verarbeitung von IT Instanzmodellen sowie eine Methode zu dessen automatisierter Erstellung ein. ETGs sind die Basis für verschiedene Arten der IT-Adaption, beispielsweise die Migration von Anwendungen von einer Ursprungs- in eine Zielumgebung.

> „The global economy runs on legacy systems – both the software and hardware – and they represent hundreds of billions of dollars in investments that enterprises have made over decades."
>
> *Aberdeen Group [Abe06]*

Die Anwendungsmigration ist von großer Bedeutung, da dadurch die signifikanten Investitionen in bestehende, erprobte Anwendungen und die Geschäftslogik erhalten werden [Hau05, Abe06]. Die Erfahrung zeigt, dass die Kosten einer Migration höchstens ein Drittel so hoch sind wie die einer Neuentwicklung [SWH10, Hug13]. Dies ist nicht verwunderlich, da Softwareprojekte durch den hohen Personalaufwand teuer und aufgrund der schweren Planbarkeit risikoreich sind. Um die Vielzahl von existierenden Anwendungen kostengünstig, schnell und kontrolliert zu migrieren, müssen die Schritte der Migration automatisiert werden [Sha12]. Der zweite Forschungsschwerpunkt ist daher die technische und automatisierte Durchführung der Anwendungsmigration bei gleichzeitiger Erhaltung deren Funktionalität.

> „What we're missing is a simple and systematic way to bring all these hundreds and thousands of apps to the cloud."
>
> *Nati Shalom auf HighScalability.com [Sha12]*

Cloud Computing verändert die IT in Firmen so schnell und gleichzeitig nachhaltig wie keine Entwicklung der IT zuvor [Ham14]. Die Cloud verspricht in erster Linie geringere Kosten, welche durch die Automatisierung, Selbstbedienung, Elastizität und ausschließliche Bezahlung für die genutzten Leistungen realisiert werden [AFG+09, Ley09, MG09, Ree09]. Dies ist der Grund, weshalb immer mehr Firmen ihre Anwendungen, mittlerweile auch ihre geschäftskritischen [Col11], in der Cloud betreiben wollen [RKM10, Cis12]. Daher betrachtet die vorliegende Arbeit die Anwendungsmigration aus einer Cloud-Perspektive, das heißt, die Cloud ist die Zielumgebung, welche zur Veranschaulichung der Konzepte sowie in den Prototypen und Fallstudien genutzt wird, aber nicht die einzige mögliche Zielumgebung.

1.1 Problemstellungen und Forschungsbeiträge

Dieser Abschnitt beschreibt die Problemstellungen, die sich aus den in der Einleitung geschilderten Zielen ergeben. Davon ausgehend werden die sechs Forschungsbeiträge der vorliegenden Arbeit vorgestellt und gezeigt, wie diese die Problemstellungen adressieren. Abbildung 1.1 veranschaulicht, wie die Forschungsbeiträge zusammenhängen: Die Migrationsmethode AROMA (Beitrag 1) nutzt das IT-Instanzmodell (Beitrag 2), das mithilfe von Beitrag 3 automatisiert erstellt wurde, als Ausgangspunkt der Migration. Aus diesem Instanzmodell wird die zu migrierende Anwendung identifiziert und isoliert (Beitrag 4). Beitrag 5 zeigt, wie die AROMA-Methode mit TOSCA umgesetzt und automatisiert werden kann. Beitrag 6 umfasst die Architektur der Prototypen, welche zur Validierung der Beiträge entwickelt wurden. Das IT-Instanzmodell (Beitrag 2), dessen Erstellung (Beitrag 3) und Verarbeitung (Beiträge 2 und 4) sowie die zugehörigen Prototypen und deren Architektur (Teile von Beitrag 6) können auch unabhängig vom Anwendungsfall Migration für die Analyse, Adaption und Optimierung der IT verwendet werden.

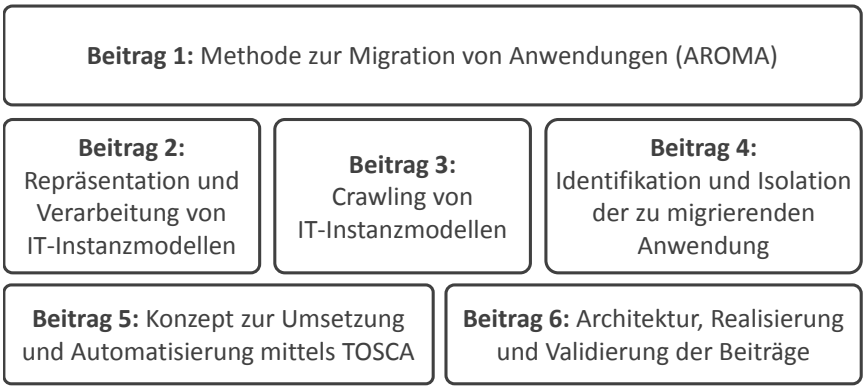

Abbildung 1.1: Übersicht der Forschungsbeiträge

1.1.1 Methode zur Migration von Anwendungen

Der erste Beitrag befasst sich mit der Problemstellung, wie die Migration einer Anwendung technisch durchgeführt werden kann. Dabei ist es erforderlich, ganzheitlich alle Komponenten und Schichten der zu migrierenden Anwendungen mitsamt aller Abhängigkeiten zu betrachten, also von der Hardware bis zur Geschäftslogik. Aufgrund der Heterogenität der Technologien, Architekturen und Komponenten bestehender Anwendungen dürfen keine besonderen Anforderungen an die Art und Weise gestellt werden, wie diese Anwendungen gebaut wurden. Um die Kosten der Migration gering zu halten, und aufgrund der großen Anzahl von potentiell zu migrierenden Anwendungen, ist die Automatisierung der Migration nötig.

> *Beitrag 1: Methode zur Migration von Anwendungen*.
> Die vorliegende Arbeit beschreibt mit AROMA (Automatisierte Migration von Anwendungen) eine Methode zur technischen Durchführung einer Anwendungsmigration aus einer Ursprungs- in eine Zielumgebung, zum Beispiel in eine Cloud. Die weiteren Forschungsbeiträge ermöglichen die Automatisierung der AROMA-Methode. Die Migration von Geschäftsprozessen wird dabei explizit durch eine Variante der Methode unterstützt.

1.1.2 Konzept zur Repräsentation und Verarbeitung von IT-Instanzmodellen

Für die Durchführung der Migration mit der AROMA-Methode (Beitrag 1) ist ein technisch detailliertes Instanzmodell der entsprechenden IT erforderlich. Dieses umfasst alle Komponenten, von der Hardware bis zu den Geschäftsprozessen, aller Anwendungen und deren Abhängigkeiten. Als Instanzmodell enthält es Laufzeitinformationen, im Gegensatz zu Architekturen oder ähnlichen Modellen, welche eine abstrakte Sichtweise repräsentieren. Um dies zu ermöglichen, wird ein generisches, erweiterbares und formales Metamodell benötigt, das ein Abbild der IT inklusive aller Komponenten, die zum Betrieb dieser Anwendung erforderlich sind, deren Abhängigkeiten und Laufzeit-

informationen darstellen kann. Zur Verarbeitung sind darauf abgestimmte Operationen nötig, die es erlauben, das Instanzmodell zu analysieren, zu transformieren und zu abstrahieren. Aufgrund der Abbildung der gesamten IT und technischen Details ist mit sehr großen Modellen zu rechnen, die entsprechend effizient verarbeitet werden müssen.

> *Beitrag 2: Konzept zur Repräsentation und Verarbeitung von IT-Instanzmodellen*.
> Die vorliegende Arbeit beschreibt mit Enterprise Topologie Graphen (ETG) ein Konzept und Metamodell für die Repräsentation von Instanzmodellen der gesamten IT. Darauf aufbauend werden Operationen zur effizienten Verarbeitung des ETGs eingeführt.

1.1.3 Crawling von IT-Instanzmodellen

Die meisten Organisationen verfügen heute über kein gesamtheitliches und aktuelles Instanzmodell ihrer IT, das alle Komponenten, von der Infrastruktur bis zu den Geschäftsprozessen, und deren Abhängigkeiten repräsentiert. Grund dafür ist, dass Anwendungen manuell verwaltet werden, sich regelmäßig weiterentwickeln und die Dokumentation, falls sie während der initialen Entwicklung und Bereitstellung der Anwendung existierte, nicht nachgezogen wurde. Dieser fehlende Einblick kann dazu führen, dass Veränderungen an der IT langsam und teuer sind oder sogar folgenreiche Fehlentscheidungen getroffen werden. Die manuelle Identifikation von Komponenten und deren Abhängigkeiten ist eine sehr zeitaufwändige, fehleranfällige und somit kostenintensive Aufgabe. Aufgrund der fehlenden Werkzeugunterstützung werden die Abhängigkeiten teilweise durch *Ausstecken* der entsprechenden Komponenten identifiziert. Auch für die Migration ist Wissen über den Aufbau der Anwendung erforderlich, insbesondere technische Details wie die Konfiguration, benötigte Dateien und Relationen verschiedener Komponenten. Die Problemstellung ist folglich: Wie kann ein gesamtheitliches, aktuelles, technisch detailliertes Instanzmodell der gesamten IT automatisiert erstellt und aktualisiert werden?

Beitrag 3: Crawling von IT-Instanzmodellen.
Die vorliegende Arbeit beschreibt eine Methode, welche das automatisierte Crawling eines gesamtheitlichen, aktuellen, technisch detaillierten Instanzmodells der IT ermöglicht, also aller Anwendungen, der für deren Betrieb nötigen Komponenten und deren Abhängigkeiten. Insbesondere werden dabei auch Geschäftsprozesse und Enterprise Service Busse unterstützt. Außerdem werden vorhandene Informationsquellen und Werkzeuge, die Teilbereiche des Instanzmodells bereits kennen, integriert.

1.1.4 Identifikation und Isolation der zu migrierenden Anwendung

Die Identifikation der für den Betrieb einer Anwendungsfunktionalität benötigten Komponenten ist ein zentraler Schritt bei der Durchführung der Migration mithilfe der AROMA-Methode (Beitrag 1). Die Problemstellung lautet dabei, wie die für den Betrieb einer Anwendung nötigen Komponenten bestimmt werden können, welche später migriert werden sollen. Dabei muss beachtet werden, wie die Anwendung mit anderen zusammenhängt, beispielsweise falls diese direkt miteinander kommunizieren oder die gleichen Komponenten verwenden. Anschließend müssen die entsprechenden Komponenten im ETG isoliert werden. Beides, Identifikation und Isolation, ist ohne ein vollständiges Instanzmodell nur schwer durchführbar.

Beitrag 4: Identifikation und Isolation der zu migrierenden Anwendung.
Die vorliegende Arbeit beschreibt ein Konzept und Algorithmen zur Bestimmung und Isolation von den Teilen der IT, die für den Betrieb einer Anwendungsfunktionalität bzw. einer Menge von Komponenten benötigt werden.

1.1.5 Konzept zur Umsetzung und Automatisierung der AROMA-Methode mittels TOSCA

Die AROMA-Methode (Beitrag 1) nutzt den Enterprise Topologie Graph (Beitrag 2), der mithilfe von Beitrag 3 erstellt wurde, als Ausgangspunkt der Migration. Nach der Identifikation und Isolation der zu migrierenden Anwendung (Beitrag 4) wird diese in eine Anwendungstopologie umgewandelt, ein Modell, das alle technischen Komponenten einer Anwendung und deren Relationen untereinander beschreibt, und so deren automatisierte Bereitstellung und Verwaltung ermöglicht. Bei der Umsetzung der Migrationsmethode sind deren automatisierte Durchführung und die Erhaltung der Anwendungsfunktionalität zentrale Anforderungen. Neben der Senkung der Kosten soll auch eine nachhaltige Verbesserung der nichtfunktionalen Eigenschaften der Anwendung erreicht werden, beispielsweise deren Portabilität. Diese spielt in diesem Kontext eine besonders wichtige Rolle, da die Bindung an einen bestimmten Anbieter (engl. *vendor lock-in*) als eines der größten Probleme von Cloud Computing angesehen wird [SJ12].

Bei der Umsetzung der AROMA-Methode mit einer konkreten Modellierungssprache für Anwendungstopologien müssen folgende Fragestellungen betrachtet werden: (i) Wie kann eine Anwendung, die von einem Teil des IT-Instanzmodells repräsentiert wird, automatisiert in eine Anwendungstopologie transformiert werden? (ii) Wie kann die zu migrierende Anwendung an die Zielumgebung angepasst werden? Dabei soll einerseits die Funktionalität der Anwendung erhalten bleiben, andererseits sollen jedoch die Eigenschaften der Zielumgebung bestmöglich genutzt werden. (iii) Wie kann es dem Benutzer ermöglicht werden, die Anwendungstopologie zu evaluieren und gegebenenfalls manuell anzupassen? (iv) Mit welchen bestehenden Konzepten kann die automatisierte Bereitstellung der Anwendungstopologie realisiert werden?

Die vorliegende Arbeit hat als Modellierungssprache für Anwendungstopologien die von OASIS standardisierte „Topology and Orchestration Specification for Cloud Applications“ (TOSCA) [BBLS12, OAS13b] gewählt und für diese die zuvor genannten Fragestellungen adressiert.

Beitrag 5: Konzept zur Umsetzung und Automatisierung der AROMA-Methode mittels TOSCA.
Die vorliegende Arbeit beschreibt ein Konzept zur Umsetzung der AROMA-Methode mit TOSCA. Dieses beinhaltet (i) eine Anwendung aus dem Instanzmodell automatisiert in eine TOSCA-Anwendungstopologie zu transformieren, (ii) eine Anwendungstopologie an die Zielumgebung anzupassen, (iii) eine Anwendungstopologie durch den Benutzer evaluieren und manuell anpassen zu lassen und (iv) diese automatisiert unter Nutzung bestehender Konzepte bereitzustellen.

1.1.6 Architektur, Realisierung und Validierung der Beiträge

Um die Forschungsbeiträge zu realisieren und sie anhand verschiedener Fallstudien zu validieren, ist ihre prototypische Implementierung nötig. Voraussetzung dafür ist die Entwicklung einer Architektur, welche die Basis für die Implementierung der Prototypen zur Verwaltung und zum Crawling von ETGs sowie die Werkzeugunterstützung der Migrationsmethode ist. Dabei ist die Erweiterbarkeit von besonderer Bedeutung, um in Zukunft neben der Migration auch weitere Anwendungsfälle realisieren zu können, beispielsweise die Sicherstellung der Regelkonformität oder der Durchführung einer Adaption der IT.

Beitrag 6: Architektur, Realisierung und Validierung der Beiträge.
Die vorliegende Arbeit stellt zur Realisierung der Forschungsbeiträge eine erweiterbare Architektur vor, die von den Prototypen ETG-Framework, ETG-Verwaltung, ETG-Crawler und dem Migrationsassistenten implementiert wird. Außerdem werden die Beiträge anhand verschiedener Fallstudien validiert.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit ist wie folgt strukturiert: Kapitel 2 betrachtet ihre grundlegenden Konzepte und Definitionen und bildet die Basis für die nachfolgenden Kapitel. Dazu werden verwandte Arbeiten im Bereich der Repräsentation von Anwendungstopologien, Instanzmodellen für die gesamte IT und deren Crawling sowie der Migration von Anwendungen vorgestellt. Dabei wird die Migration von Anwendungen unter dem Aspekt der automatisierten technischen Migrationsdurchführung betrachtet, insbesondere mit der Cloud als Zielumgebung der Migration.

Kapitel 3 stellt die in dieser Arbeit entwickelte AROMA-Methode zur Migration von Anwendungen vor. Nach Betrachtung der identifizierten Anforderungen wird detailliert auf die sieben Schritte der Methode eingegangen. Davon ausgehend werden verschiedene Varianten der AROMA-Methode beschrieben und im Rahmen der Diskussion für die Migration geeignete Anwendungen und Zielumgebungen evaluiert.

Kapitel 4 führt mit Enterprise Topologie Graphen (ETG) ein Konzept und Metamodell zur Repräsentation und Verarbeitung von IT-Instanzmodellen ein. Nach einem informellen Überblick folgt die formale Definition von ETGs. Anschließend werden verschiedene Operationen auf ETGs eingeführt, die im Verlauf dieser Arbeit verwendet werden.

Kapitel 5 beschäftigt sich mit dem Crawling von IT-Instanzmodellen, die durch den in Kapitel 4 eingeführten ETG repräsentiert werden. Dazu werden eine Plugin-basierte Methode und ihre Architektur, Rollen, Algorithmen und Qualitätssicherung vorgestellt. Davon ausgehend wird detailliert die Arbeitsweise ausgewählter Plugins betrachtet und die Entwicklung von Crawler-Plugins behandelt. Das Kapitel schließt mit einer Evaluation und Validierung der vorgestellten Methode und Algorithmen sowie einer Diskussion der Ergebnisse.

Kapitel 6 stellt das Konzept zur Umsetzung der AROMA-Methode mit TOSCA als Sprache zur Beschreibung von Anwendungstopologien vor. Dieses beinhaltet die Transformation, Adaption, Evaluation, Paketierung und Bereitstellung der zu migrierenden Anwendung mit TOSCA.

Kapitel 7 beschreibt die Architektur der Prototypen zur Realisierung der entwickelten Konzepte und Methoden. Außerdem werden ausgewählte technische Aspekte der Implementierung motiviert und diskutiert.

Kapitel 8 validiert die vorgestellten Methoden, Konzepte und Prototypen in verschiedenen Fallstudien und evaluiert diese anhand ausgewählter Kriterien. Die vorliegende Arbeit wird abgeschlossen durch Kapitel 9, welches die Forschungsbeiträge zusammenfasst, diskutiert und einen Ausblick auf zukünftige Arbeiten gibt.

1.3 Veröffentlichungen

Nachfolgende begutachtete Veröffentlichungen sind bereits aus der Forschung im Rahmen dieses Promotionsvorhabens hervorgegangen:

1. [BBKL14a] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann. Advanced Web Services, Kapitel **TOSCA: Portable Automated Deployment and Management of Cloud Application**s, S. 527–549. Springer, 2014.

2. [BBKL14b] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann. **Migration of enterprise applications to the cloud**. it – Information Technology, Special Issue: Architecture of Web Application, 56(3):106–111, 2014.

3. [BBH+13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, S. Wagner. **OpenTOSCA – A Runtime for TOSCA-based Cloud Applications**. In Proceedings of the 11th International Conference on Service-Oriented Computing, S. 692–695. 2013.

4. [BBKL13a] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann. **Automated Discovery and Maintenance of Enterprise Topology Graphs**. In Proceedings of the 6th IEEE International Conference on Service Oriented Computing & Applications, S. 126–134. 2013.

5. [BBK+13] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann, A. Weiß. **Improve Resource-Sharing through Functionality-Preserving Merge**

of Cloud Application Topologies. In Proceedings of the 3rd International Conference on Cloud Computing and Service Science, S. 96–103. 2013.

6. [BBLS12] T. Binz, G. Breiter, F. Leymann, T. Spatzier. **Portable Cloud Services Using TOSCA**. IEEE Internet Computing, 16(03):80–85, 2012.

7. [BFL+12] T. Binz, C. Fehling, F. Leymann, A. Nowak, D. Schumm. **Formalizing the Cloud through Enterprise Topology Graphs**. In Proceedings of the 5th IEEE International Conference on Cloud Computing, S. 742–749. 2012.

8. [BLNS12] T. Binz, F. Leymann, A. Nowak, D. Schumm. **Improving the Manageability of Enterprise Topologies Through Segmentation, Graph Transformation, and Analysis Strategies**. In Proceedings of the 16th IEEE International Enterprise Distributed Object Computing Conference, S. 61–70. 2012.

9. [BLS11] T. Binz, F. Leymann, D. Schumm. **CMotion: A Framework for Migration of Applications into and between Clouds**. In Proceedings of the 9th IEEE International Conference on Service-Oriented Computing and Applications, S. 225–228. 2011.

Weitere begutachtete Veröffentlichungen des Autors, die eine wichtige Rolle in dieser Arbeit spielen oder deren Forschungsbeiträge einsetzen:

1. [BBKL14c] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann. **Automating Cloud Application Management Using Management Idioms**. In Proceedings of the 6th International Conferences on Pervasive Patterns and Applications, S. 60–69. 2014.

2. [BBK+14a] U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann, J. Wettinger. **Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA**. In Proceedings of the IEEE International Conference on Cloud Engineering, S. 87–96. 2014.

3. [BBK+14b] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, M. Wieland. **Context-aware Cloud Application Management**. In Proceedings of the 4th International Conference on Cloud Computing and Services Science, S. 499–509. 2014.

4. [HBBL14] P. Hirmer, U. Breitenbücher, T. Binz, F. Leymann. **Automatic Topology Completion of TOSCA-based Cloud Applications**. In Proceedings of the Informatik 2014, Band P-232 von LNI, S. 247–258. 2014.

5. [ABLS13] V. Andrikopoulos, T. Binz, F. Leymann, S. Strauch. **How to adapt applications for the Cloud environment**. Computing, Springer, 95(6):493–535, 2013.

6. [BBKL13b] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann. **Pattern-based Runtime Management of Composite Cloud Applications**. In Proceedings of the 3rd International Conference on Cloud Computing and Service Science. 2013.

7. [NBLU13] A. Nowak, T. Binz, F. Leymann, N. Urbach. **Determining Power Consumption of Business Processes and their Activities to Enable Green Business Process Reengineering**. In Proceedings of the 17th IEEE International Enterprise Distributed Object Computing Conference, S. 259–266. 2013.

8. [WBB+13] J. Wettinger, M. Behrendt, T. Binz, U. Breitenbücher, G. Breiter, F. Leymann, S. Moser, I. Schwertle, T. Spatzier. **Integrating Configuration Management with Model-Driven Cloud Management Based on TOSCA**. In Proceedings of the 3rd International Conference on Cloud Computing and Service Science, S. 437–446. 2013.

9. [NBF+12] A. Nowak, T. Binz, C. Fehling, O. Kopp, F. Leymann, S. Wagner. **Pattern-driven Green Adaptation of Process-based Applications and their Runtime Infrastructure**. Computing, S. 463–487, 2012.

Begutachtete Konferenz- und Workshopbeiträge sowie Demonstratoren, welche das in Abschnitt 6.2 vorgestellte OpenTOSCA-Ökosystem behandeln:

1. [BBKL14e] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann. **Vinothek - A Self-Service Portal for TOSCA**. In N. Herzberg, M. Kunze, Herausgeber, Proceedings of the 6th Central-European Workshop on Services and their Composition, Band 1140 von CEUR Workshop Proceedings, S. 69–72. 2014.

2. [BBH+13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, S. Wagner. **OpenTOSCA – A Runtime for TOSCA-based Cloud Applications**. In Proceedings of the 11th International Conference on Service-Oriented Computing, S. 692–695. 2013.

3. [KBBL13] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann. **Winery – A Modeling Tool for TOSCA-based Cloud Applications**. In Proceedings of the 11th International Conference on Service-Oriented Computing, S. 700–704. 2013.

4. [BBK+12] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, D. Schumm. **Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA**. In Proceedings of the 20th International Conference on Cooperative Information Systems, S. 416–424. 2012.

5. [KBBL12] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann. **BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications**. In Proceedings of the 4th Business Process Model and Notation Workshop, S. 38–52. 2012.

KAPITEL 2

GRUNDLAGEN UND VERWANDTE ARBEITEN

Dieses Kapitel führt die grundlegenden Konzepte und Definitionen der vorliegenden Arbeit ein und diskutiert davon ausgehend den aktuellen Stand der Wissenschaft und Technik in diesen Bereichen. Nach einem kurzen Überblick über Service-orientierte Architekturen in Abschnitt 2.1 führt Abschnitt 2.2 Anwendungen und deren Bestandteile ein. Abschnitt 2.3 beschreibt Cloud Computing, definiert dessen Eigenschaften und betrachtet die Besonderheiten von Cloud-Anwendungen. Verschiedene Spezifikationen zur Beschreibung von Anwendungstopologien werden in Abschnitt 2.4 erörtert. Verwandte Arbeiten zu IT-Instanzmodellen und deren Erstellung werden in Abschnitt 2.5 bzw. Abschnitt 2.6 vorgestellt. Schlussendlich betrachtet Abschnitt 2.7 den aktuellen Stand der Wissenschaft und Technik der Migration von Anwendungen. Dabei wird zwischen Migrationsdurchführung, Vorgehensmodellen und Strategien zur Migration unterschieden.

2.1 Service-orientierte Architekturen

„Dienste“ (engl. *services*) kapseln eine bestimmte Funktionalität und stellen diese nach außen zur Verfügung [Pap03, KBS04]. Dabei verstecken Dienste ihre Implementierung hinter einer präzise definierten Schnittstelle und sind für den Nutzer immer verfügbar (engl. *always-on*) [WCL+05]. „Service-orientierte Architekturen“ (SOA) [Pap03] sind ein heute weitverbreiteter Architekturstil, der darauf abzielt, jegliche Funktionalität einer Anwendung als Dienst zu kapseln, zur Wiederverwendung anzubieten und miteinander zu kombinieren. In einer SOA sind Dienste lose gekoppelt und können dadurch flexibel kombiniert werden, um eine neue Funktionalität zu kreieren, die wiederum als Dienst angeboten wird (*rekursives Aggregationsmodell* [WCL+05]). Basierend darauf werden Geschäftsprozesse heutzutage durch die Kombination vieler Dienste realisiert [Pap03]. Der Geschäftsprozess wird dabei durch einen Graphen modelliert, der beschreibt, welche Dienste wie orchestriert, also kombiniert, werden sollen [LR00, OAS07].

Alle Dienste werden in einem Verzeichnis publiziert und können dadurch von Konsumenten gefunden und genutzt werden. Diese Aufgaben werden von einem „Enterprise Service Bus“ (ESB) übernommen, welcher die zentrale Komponente einer Service-orientierten Architektur darstellt [Cha04]. Alle Aufrufe der registrierten Dienste durchlaufen den ESB, wobei der ESB den Dienst anhand verschiedenster Kriterien auswählt und den Aufruf möglicherweise auch transformiert.

„Webservices“ sind interoperable Dienste, die über eine Webadresse angeboten und über ein Netzwerk konsumiert werden [HB04]. Sie stellen basierend auf den sogenannten „WS-* Standards“ [WCL+05] eine Möglichkeit dar, eine Service-orientierte Architektur zu realisieren. Zu dieser Familie von Standards gehört beispielsweise „WSDL“ (Web Services Description Language) [CCMW01] zur Beschreibung der Schnittstelle von Webservices oder „WS-Policy“ [W3C07] zur Beschreibung und Aushandlung nichtfunktionaler Anforderungen. Die ausführbare Orchestrierung von Webservices wird typischerweise mit „BPEL“ (Business Process Execution Language) [OAS07] oder „BPMN 2.0“ (Business Process Model and Notation) [OMG11a] rea-

lisiert [Ley10]. Alle Funktionalitäten als über das Netzwerk zugängliche Dienste anzubieten, ist auch ein zentrales Konzept von Cloud Computing [Ley09], das in Abschnitt 2.3 eingeführt wird.

2.2 Anwendungen

In der Umgangssprache wird eine *Anwendung* (engl. *application*) oft mit einem Programm oder einer Software in Verbindung gebracht [Gmb13], die eine bestimmte Funktionalität implementiert und eine Benutzeroberfläche anbietet. Technisch betrachtet sind Anwendungen aber keine monolithischen Blöcke, sondern werden durch eine Reihe unterschiedlichster physikalischer oder virtueller Komponenten realisiert [MFKL10]. Dazu gehören zum Beispiel die Laufzeitumgebung, das Betriebssystem, die Datenbank, der Server und die Netzwerkkomponenten. Unter dem Begriff „Anwendung“ werden in dieser Arbeit daher alle Komponenten zusammengefasst, die zum Betrieb deren Funktionalität nötig sind. Anwendungen, die eine komplexe Geschäftslogik auf großen Datenmengen ausführen, auf die parallel von vielen Nutzern zugegriffen wird und die stark integriert mit anderen Anwendungen sind, bezeichnet Fowler [Fow02] als „Geschäftsanwendungen“. Die vorliegende Arbeit betrachtet alle Arten von Anwendungen, insbesondere auch Geschäftsanwendungen (siehe Abschnitt 3.4.1), weshalb im Folgenden von Anwendung gesprochen wird.

Definition 2.1 (Anwendung – informell)**.** Eine Anwendung ist die Menge aller Komponenten, die zum Betrieb der Funktionalität der Anwendung nötig sind.

Um die Komplexität beherrschen zu können, werden Anwendungen in Schichten aufgeteilt. Tanenbaum und van Steen [TVS02] unterscheiden die drei Schichten: (i) Benutzeroberfläche, (ii) Verarbeitung und (iii) Daten. Genauso Fowler [Fow02], der drei Schichten der Architektur von Geschäftsanwendungen definiert: (i) Darstellung, (ii) Geschäftslogik und (iii) Daten. Im Cloud Computing wird zwischen drei Arten von Diensten unterschieden [YBDS08, MG09]: (i) Software (engl. *Software as a service*), (ii) Platt-

form (engl. *Platform as a service*) und (iii) Infrastruktur (engl. *Infrastructure as a service*). Diese wurden in verschiedenen Veröffentlichungen um Geschäftsprozesse erweitert (engl. *Business process as a service* [RKM10, BSB11] oder *Composition as a service* [Ley09, RLM+09]).
Ausgehend davon werden drei Arten von Bestandteilen definiert, um die Komponenten einer Anwendung zu klassifizieren:

Bestandteil 1: Anwendungsfunktionalität. Darunter wird Software verstanden, welche die Funktionalität einer Anwendung implementiert. Anwendungsfunktionalität wird beispielsweise von Diensten im Sinne von SOA (vgl. Abschnitt 2.1) implementiert, die ihre Funktionalität ausschließlich über eine Schnittstelle zur Verfügung stellen. Ein Geschäftsprozess wiederum orchestriert verschiedene Dienste zu einer höherwertigen Funktionalität, die er als Dienst nach außen anbietet (vgl. Abschnitt 2.1). Zwischen Diensten und ihrer Komposition (d.h. Geschäftsprozessen) wird dabei nicht unterschieden, da beide Anwendungsfunktionalität implementieren. Eine andere Art ist das Anwendungsprogramm oder eine Webseite, welche auch eine Benutzeroberfläche für Endbenutzer anbieten.

Bestandteil 2: Middleware. Anwendungsfunktionalität, insbesondere von verteilten und komplexen Geschäftsanwendungen, läuft üblicherweise nicht direkt auf dem Betriebssystem, sondern nutzt eine Reihe von Abstraktionen und unterstützenden Funktionalitäten [Cha99, Emm00]. Dazu gehören beispielsweise die Verwaltung von Clustern, das Sicherstellen von Service Level Agreements, die Persistierung von Daten, die Durchführung von Transaktionen, der Transport von Nachrichten und vieles mehr [Emm00]. Diese unterstützenden Funktionalitäten werden von Middleware-Komponenten angeboten, zum Beispiel einem Application Server, Datenbank Management System (DBMS), Workflow Management System (WFMS), Enterprise Service Bus (ESB) oder Message Queueing System. Diese implementieren keine Anwendungsfunktionalität, sondern bieten eine Abstraktion, die dem Entwickler hilft, Anwendungsfunktionalität zu implementieren, ohne die zuvor genannten, nicht anwendungsspezifischen Funktionalitäten selbst realisieren zu müssen. Neben den oben genannten gibt es *neue Arten* von Middleware, etwa zur Analyse großer Datenmengen (Big Data), wie zum

Beispiel „MapReduce“ [DG08]. Im Cloud Computing wird die Middleware üblicherweise als *Plattform* bezeichnet und umfasst alle Komponenten *zwischen* der Anwendungsfunktionalität und dem Betriebssystem.[1]

Bestandteil 3: Infrastruktur. Unter Infrastruktur versteht man alle Komponenten, welche die drei Grundbausteine Rechenleistung, Speicher und Netzwerkkommunikation anbieten [MG09]. Üblicherweise werden diese Grundbausteine heute virtualisiert genutzt [BYV+09], um Flexibilität zu gewinnen und die physikalische Hardware optimal auszulasten [Hum06]. Bei der Nutzung von virtuellen Maschinen (VM) in der Cloud wird das Betriebssystem üblicherweise auch zur Infrastruktur gezählt [MG09]. Zwischen virtuellen und physikalischen Infrastrukturkomponenten wird im Folgenden nicht unterschieden.

2.3 Cloud Computing

Cloud Computing setzt unter dem Stichwort „Everything as a service“ die Serviceorientierung konsequent fort, indem nicht nur Anwendungslogik, sondern jede IT-Komponente als Dienst zur Verfügung gestellt wird [PBA+08]. Es wird prinzipiell also jede Art von IT als Dienst angeboten, auch Netzwerk-, Hardware- und Middleware-Komponenten. Somit erlaubt es Cloud Computing einmalige IT-Investitionen (engl. *capital expenses*) durch die nutzungsabhängige Bezahlung eines externen Anbieters zu ersetzen (engl. *operational expenses*) [AFG+09, Ley09]. Die daraus resultierende Kostenreduktion ist die Motivation für die Nutzung der Cloud. Darauf aufbauend können Anwendungen durch Nutzung der in Abschnitt 2.3.1 definierten Cloud-Eigenschaften (Selbstbedienung, Elastizität, Nutzungsabhängige Bezahlung und Automatisierung) weitere Kostensenkungen erzielen. Die Cloud ist nur eine Zielumgebung für die Migration von Anwendungen und wurde in dieser Arbeit als exemplarische Zielumgebung gewählt.

[1] Aufgrund der großen Bandbreite dieser Gruppe gibt es Vorschläge, diese weiter aufzuteilen: Johan den Haan [Joh13] beispielsweise unterscheidet „Foundational PaaS“ (Betreiben von Anwendungspaketen mit Fokus auf die Anwendungsinfrastruktur), „PaaS“ (Betreiben der Anwendung basierend auf Code) und „Model-Driven PaaS“ (domänenspezifische Sprachen zur abstrakten, nicht technischen Beschreibung der Anwendung).

Nach der Definition der Cloud-Eigenschaften in Abschnitt 2.3.1 diskutiert Abschnitt 2.3.2 die verschiedenen Arten von Cloud-Anwendungen.

2.3.1 Cloud-Eigenschaften

Cloud Computing wird heute hauptsächlich durch eine Menge von Eigenschaften definiert, wobei in der Literatur [AFG+09, Ley09, MG09, Ree09, Nie11, VBB11, FLR+14] noch keine Einigkeit über ihre genaue Zusammenstellung besteht. Dieser Abschnitt definiert die vier Eigenschaften von Cloud Computing, welche von dieser Arbeit als die fundamentalen Cloud-Eigenschaften angesehen werden.

Cloud-Eigenschaft 1 (CE-1): Selbstbedienung. Der Nutzer kann jederzeit, abhängig von seinen Bedürfnissen, Dienste und Ressourcen nachfragen oder wieder freigeben [Nie11, VBB11]. Diese werden ihm automatisiert und direkt, ohne manuelles Eingreifen von Seiten des Anbieters, zur Verfügung gestellt [MG09].

Cloud-Eigenschaft 2 (CE-2): Elastizität. Der Nutzer kann den Umfang der konsumierten Ressourcen jederzeit erhöhen oder vermindern [MG09, Nie11, VBB11, FLR+14]. Auf diese Anfragen reagiert der Anbieter schnell [MG09] und ohne Einschränkungen des Dienstes, zum Beispiel einer vorübergehenden Unterbrechung der Verfügbarkeit. Die Anzahl der nachgefragten Ressourcen kann beliebig hoch sein, so dass für den Nutzer der Eindruck unbegrenzter Ressourcen entsteht [AFG+09, Ley09, AFG+10, DGST11].

Cloud-Eigenschaft 3 (CE-3): Nutzungsabhängige Bezahlung. Die (sich aufgrund der Elastizität ständig ändernde) Nutzung des Dienstes wird gemessen und vom Nutzer auch ausschließlich diese bezahlt [AFG+09, Ley09, Ree09, AFG+10, Nie11, VBB11]. Dabei wird die Nutzung in einer dem Dienst angemessenen Abstraktion gemessen [MG09, BGRV12], zum Beispiel Nutzer pro Monat anstatt Lizenzen und Server.

Cloud-Eigenschaft 4 (CE-4): Automatisierung. Die Bereitstellung und Verwaltung der Ressourcen und Dienste für den Nutzer erfolgt automatisiert [MG09, FLR+14]. Die Automatisierung ist ein Kernaspekt von Cloud Computing, welcher eng mit der Bereitstellung als Dienst verbunden ist. Diese

Eigenschaft wird insbesondere durch CE-1 und CE-2 impliziert, die sonst nicht realisierbar wären. Auch die Senkung der Kosten, die zentrale Motivation für die Nutzung von Cloud Computing, kann nur durch das Entfernen der manuellen Aktivitäten aus dem Prozess realisiert werden.

Tabelle 2.1 zeigt, dass die gewählten Eigenschaften von der Literatur gedeckt, jedoch in dieser Kombination in keiner Quelle gemeinsam vorkommen. Auf der y-Achse sind die verschiedenen Literaturquellen aufgelistet, auf der x-Achse die zuvor definierten Cloud-Eigenschaften. In den Zellen markiert ✓, dass eine Eigenschaft von der Literaturquelle genannt wird, und ✗, dass sie nicht genannt wird.

	Cloud-Eigenschaft			
	Selbstbedienung	Elastizität	Nutzungsabhängige Bezahlung	Automatisierung
Mell und Grance [MG09]	✓	✓	✓	(✓)[1]
Leymann [Ley09]	✗	✓	✓	✗
Armbrust et al. [AFG+09]	✗	✓	✓	✗
Fehling et al. [FLR+14]	✗	✓	✗	✓
Nielsen [Nie11]	✓	✓	(✓)[2]	✗
Voorsluys et al. [VBB11]	✓	✓	✓	(✓)[3]
Reese [Ree09]	✗	✗	✓	✗

Tabelle 2.1: Nennung der definierten Cloud-Eigenschaften in der Literatur

[1] Wird impliziert durch die Definition von „On-demand self-service“: „[...] automatically without requiring human interaction with each service's provider“ [MG09].

[2] Die von Nielsen [Nie11] genannte Skalierbarkeit ist ein Aspekt von Elastizität, jedoch legt zweitere den Schwerpunkt darauf, *hoch* und *runter* skalieren zu können, also *zu atmen*.

[3] Enthalten in der Beschreibung von „Self-Service“: „[...] without intervention of human operators“ [VBB11]. Als Beleg wird Mell und Grance [MG09] referenziert.

Die Literatur führt verschiedene weitere Eigenschaften an, welche jedoch bewusst nicht aufgenommen wurden: (i) Mehrmandantenfähigkeit [FWS10, Wil12, SALM12] ist die gemeinsame Nutzung einer Komponente durch verschiedene Mandanten (engl. *tenants*) [GSH+07, MUTL09], welche zu einer deutlichen Reduzierung der Kosten führt. Damit ist Mehrmandantenfähigkeit ein in der Cloud oft angewendetes Implementierungsdetail, jedoch keine grundlegende Cloud-Eigenschaft. Die Mehrmandantenfähigkeit einer existierenden Anwendung wird durch ihre Migration nicht verändert, das heißt, war diese vor der Migration mehrmandantenfähig, ist sie dies auch nach der Migration. Anderenfalls ist die Mehrmandantenfähigkeit eine neue Anforderung, die in einem Projekt zur Weiterentwicklung der Anwendung adressiert werden muss. (ii) Verteilung [FWS10, FLR+14] und horizontale Skalierbarkeit [Wil12] beschreiben Möglichkeiten zur Umsetzung der Eigenschaft CE-2 („Elastizität“), die dabei helfen, Kosteneinsparungen zu realisieren. Dies ist jedoch auch ein Implementierungsdetail und stellt keine grundlegende Cloud-Eigenschaft dar.

2.3.2 Cloud-Anwendungen

Anwendungen werden primär zur Reduktion der Kosten in die Cloud migriert. Darauf aufbauend ist sekundär auch der Grad der Nutzung (engl. *exploitation*) der Cloud-Eigenschaften wichtig. Inwiefern eine Anwendung von der Cloud profitiert, wird in der Literatur unter anderem anhand der Unterscheidung in zwei Klassen ausgedrückt: (i) „Cloud-native“ [FWS10, ABLS13, JAP13] oder „cloud-optimized“ [FH11a] Anwendungen wurden speziell für die Cloud entwickelt, um deren Eigenschaften optimal zu nutzen. (ii) „Cloud-enabled“ [FWS10, ABLS13, JAP13], „cloud-immigrant“ [GGJGT+12] oder „cloud-aligned“ [FH11a] Anwendungen existieren bereits und wurden so adaptiert, dass sie in der Cloud betrieben werden können. Diese Klassifikation bringt zum Ausdruck, dass Anwendungen, die speziell für die Cloud entwickelt wurden, deren Eigenschaften besser nutzen können als Anwendungen, welche nur für die Cloud adaptiert wurden [FWS10, ABLS13].

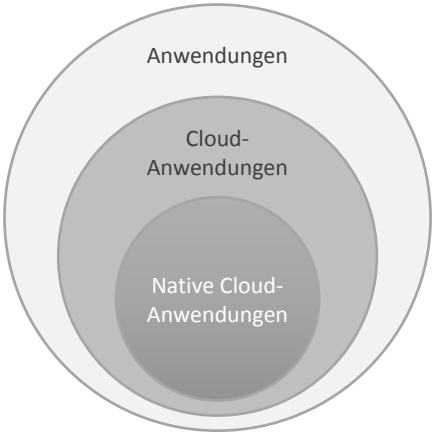

Abbildung 2.1: Relation der verschiedenen Mengen von Anwendungen

Die vorliegende Arbeit unterscheidet daher zwischen Anwendungen (vgl. Definition 2.1), Cloud-Anwendungen und nativen Cloud-Anwendungen, deren Zusammenhang in Abbildung 2.1 dargestellt wird:

Definition 2.2 (Cloud-Anwendung – informell)**.** Eine *Cloud-Anwendung* ist eine Anwendung, die in der Cloud betrieben wird. Dabei profitiert die Anwendung von der Kostenreduktion und den Cloud-Eigenschaften allgemein, nutzt diese aber nicht zwangsläufig bestmöglich.

Definition 2.3 (Native Cloud-Anwendung – informell)**.** *Native Cloud-Anwendungen* sind in der Cloud *heimisch*. Das heißt, sie nutzen die Eigenschaften der Cloud bestmöglich [FWS10, ABLS13], da sie die Möglichkeiten und Konzepte des Cloud Computings konsequent anwenden [FWS10, Wil12].

Im Gegensatz zu manchen Ansätzen in der Literatur [FWS10, FH11a] schließt die vorliegende Arbeit jedoch nicht aus, dass eine Anwendung durch die Migration in und Anpassung an die Cloud eine native Cloud-Anwendung wird. Wenn möglich sollen die Anwendungen durch die Migration native Cloud-Anwendungen werden.

2.4 Anwendungstopologien

Eine Anwendungstopologie ist ein Modell, welches alle technischen Komponenten einer Anwendung und deren Relationen untereinander beschreibt. Als formale und maschinenlesbare Modelle ermöglichen Anwendungstopologien ein gemeinsames Verständnis der Anwendung zwischen verschiedenen Rollen und Werkzeugen sowie die Automatisierung der Bereitstellung, Verwaltung und Außerbetriebnahme der Anwendung. Wichtigster Aspekt ist dabei die Aufteilung der Anwendung in Komponenten, die auch ein zentrales Konzept der Modularisierung [Gra86] und Serviceorientierung (vgl. Abschnitt 2.1) ist.

Definition 2.4 (Anwendungstopologie – informell)**.** Eine Anwendungstopologie ist ein formales und maschinenlesbares Modell, das alle technischen Komponenten einer modularen Anwendung und deren Relationen untereinander beschreibt, um beispielsweise deren automatisierte Bereitstellung und Verwaltung zu ermöglichen. Die Anwendungstopologie enthält auch die Konfiguration aller Komponenten und benötigte Artefakte, beispielsweise Softwarepakete.

Anwendungstopologien unterscheiden sich von Architekturmodellen darin, dass erstere deutlich detaillierter sind und technische Informationen für Bereitstellung und Betrieb der Anwendung enthalten. Architekturen werden normalerweise zur Kommunikation und Dokumentation einer abstrakten Systemsicht verwendet, beispielsweise „Fundamental Modeling Concepts“ (FMC) [KW03], nicht jedoch für technische Details oder zur Ausführung.

Die Komponenten mit Anwendungsfunktionalität, welche die eigentliche Logik der Anwendung implementieren, sind oben in der Anwendungstopologie zu finden (vgl. Abschnitt 2.2). Umso weiter unten Komponenten in der Anwendungstopologie sind, desto generischer, standardisierter und damit auch austauschbarer sind sie [BLS11, ASLW14]. Komponenten, die keine Anwendungsfunktionalität repräsentieren, betreiben üblicherweise die Komponenten mit Anwendungsfunktionalität.

Dieser Abschnitt gibt im Folgenden einen Überblick über Anwendungstopologien, einerseits aus dem Blickwinkel der komponentenbasierten Softwareentwicklung in Abschnitt 2.4.1 und andererseits in Abschnitt 2.4.2 mit Blick auf den Stand der Wissenschaft und Technik zu Modellen, welche die automatisierte Bereitstellung von Anwendungen ermöglichen. Als konkretes Beispiel stellt Abschnitt 2.4.3 den OASIS-Standard TOSCA [OAS13b] vor.

2.4.1 Komponentenbasierte Softwareentwicklung

In der Softwareentwicklung wird die Modularisierung, das heißt die Aufteilung der Anwendung in Komponenten, zur Verbesserung der Flexibilität, Wiederverwendbarkeit und Verständlichkeit angewendet [Mie10]. Modelle aus der Softwareentwicklung repräsentieren jedoch überwiegend funktionale Aspekte und nicht die technischen Details, die zum Betrieb von Anwendungen nötig sind. Ein Metamodell aus der Softwareentwicklung, welches Anwendungstopologien nahekommt, sind „UML Deployment Diagramme“ [OMG11b]. Diese beschreiben die Verteilung von Artefakten, zum Beispiel die Java-Implementierung eines Dienstes, auf Knoten, beispielsweise einem JEE-Server. Ähnlich wie Architekturen werden diese Diagramme jedoch zur Kommunikation verwendet, nicht zur automatisierten Bereitstellung.

2.4.2 Verfahren zur automatisierten Bereitstellung von Anwendungen

Dieser Abschnitt stellt Verfahren vor, welche ein Modell der Anwendung nutzen, um diese automatisiert bereitzustellen. „CloudML“ [BMM12] ist eine Modellierungssprache, um Infrastrukturkomponenten abstrakt zu modellieren. Diese Modelle können mit dem dazugehörigen Werkzeug bei verschiedenen Cloud-Anbietern bereitgestellt werden. Die automatisierte Bereitstellung von zusammengesetzten Anwendungen (engl. *composite applications*) mithilfe einer Anwendungstopologie, welche über die Infrastruktur hinausgeht, beschreiben beispielsweise Mietzner und Leymann [ML10]. Dabei besteht eine Anwendungstopologie aus Komponenten, die mit einer *deployedOn*-Relation verbunden werden, und aus Implementierungen der

Komponenten, beispielsweise Images oder ZIP-Dateien. Darüber hinaus wird die Anwendung mit einem Variabilitätsmodell verbunden, welches die Konfigurationsmöglichkeiten der Anwendung formalisiert.

„DevOps“ [HM11] führt die Rollen von Entwicklern (engl. *developers*) und Administratoren (engl. *operators*) zusammen, so dass die Entwickler ihre Anwendungen selbst beschreiben, bereitstellen und betreiben. Die meisten DevOps-Werkzeuge nutzen jedoch keine explizite Anwendungstopologie, da Komponenten nur Abhängigkeiten zu anderen Komponenten definieren, zum Beispiel „Chef“ [Che14a], „Puppet“ [Pup14] oder „CloudFormation“ [Ama14a]. Diesen Abhängigkeiten folgend werden die Komponenten der Anwendung durch die entsprechenden DevOps-Werkzeuge automatisch bereitgestellt. Ein Modell der gesamten Anwendung liegt dabei nicht vor, da die Relationen nur implizit enthalten sind. In „JuJu“ [Can14] beschreibt jede Komponente sowohl die Funktionen, welche von anderen Komponenten benötigt werden (z.B. eine Datenbank), als auch die von ihr angebotenen Funktionen (z.B. Einrichtung und Betrieb einer Datenbank). Im Gegensatz zur ausschließlichen Beschreibung von Abhängigkeiten in anderen Ansätzen vereinfacht dies die Komposition von Komponenten, weil die Möglichkeiten zur Komposition explizit im Modell repräsentiert werden.

Falls die in der Anwendungstopologie beschriebene Anwendung von einer entsprechenden Laufzeitumgebung automatisch betrieben wird, spricht man heute teilweise von „Software defined Environment“ (SDE) [BBG+14] oder, nachdem ein gewisser Hype rund um diese Begrifflichkeit entstanden ist, auch von „Software defined Everything“ [Mat13]. Diese unter dem Namen „Software defined Networks“ (SDN) ursprünglich aus dem Netzwerkbereich stammende Bezeichnung [McK09] betrachtet die IT als ein flexibles Modell, insbesondere auch die Hardware, welches durch Software automatisiert betrieben wird. Auch wenn sich eine klare Definition von SDE noch nicht herauskristallisiert, beschreibt es im Kern die konsequente Umsetzung der Konzepte des Cloud Computings (vgl. Abschnitt 2.3).

2.4.3 Topology and Orchestration Specification for Cloud Applications

Der OASIS-Standard „Topology and Orchestration Specification for Cloud Applications“ (TOSCA) [OAS13b, BBLS12, BBKL14a] hat die Ziele, die Bereitstellung und Verwaltung von Anwendungen zu automatisieren, Portabilität und Interoperabilität zu gewährleisten und ein anbieterunabhängiges Ökosystem aufzubauen. TOSCA beschreibt zwei Teile einer Anwendung: (i) die Anwendungstopologie und (ii) die Orchestrierung der von den Komponenten und Relationen der Anwendungstopologie angebotenen Operationen. Dabei erlaubt es TOSCA, alle der in Abschnitt 2.2 definierten Bestandteile einer Anwendung zu repräsentieren. Dieser Abschnitt stellt die zentralen Konzepte von TOSCA vor und verwendet dabei im Folgenden die englischen Originalnamen des Standards.

2.4.3.1 Anwendungstopologie in TOSCA

Eine Anwendungstopologie in TOSCA ist ein Graph bestehend aus Nodes (typisierte Komponenten) und Relationships (typisierte und gerichtete Relationen). NodeTypes beschreiben unterschiedliche Arten von Komponenten und RelationshipTypes, die unterschiedlichen Arten, wie diese Komponenten voneinander abhängen, aufeinander aufbauen oder miteinander verbunden sind. Alle Typen in TOSCA sind unabhängig von der Anwendungstopologie, in der sie verwendet werden, und somit in verschiedenen Anwendungen wiederverwendbar. TOSCA sieht ein offenes Typsystem vor, das heißt, es kann erweitert werden und schränkt die Arten von Typen nicht ein. Dies beinhaltet auch, dass Typen von anderen Typen erben können, um diese zu erweitern oder Teile deren Definition zu überschreiben. Um die Semantik der Nodes und Relationships eines Typs zu beschreiben, definiert TOSCA unter anderem die Properties, DeploymentArtifacts und Operations eines Typs. Über Properties werden Nodes und Relationships konfiguriert, beispielsweise durch Setzen der Version oder des Benutzernamens. DeploymentArtifacts repräsentieren die Implementierung von Nodes, beispielsweise das Image einer virtuellen Maschine oder ein Java Web Archive. Operations beschreiben Ver-

waltungsaktivitäten, die auf der Node oder Relationship ausgeführt werden können und in Interfaces gruppiert werden. Dabei wird nur die Schnittstelle beschrieben (Eingabe- und Ausgabeparameter sowie deren Datenformat), nicht wie diese Operations implementiert werden. NodeTypeImplementations bzw. RelationshipTypeImplementations stellen Implementierungen für die in den Typen definierten Operations bereit. Beispiele für solche ImplementationArtifacts sind ein Script, eine durch DevOps-Werkzeuge ausführbare Beschreibung [WBB+13] oder ein in Java implementierter Dienst. TOSCA erlaubt es, mehrere Implementations, DeploymentArtifacts und ImplementationArtifacts anzugeben, von denen zur Laufzeit eines auswählt wird. So kann beispielsweise das Image einer virtuellen Maschine in mehreren Formaten für verschiedene Infrastruktur-Clouds definiert und so die Portabilität einer Anwendungstopologie erhöht werden.

In einem TopologyTemplate, was im Kontext dieser Arbeit eine Anwendungstopologie ist, werden NodeTypes als NodeTemplates und RelationshipTypes als RelationshipTemplates modelliert und für die Verwendung in der entsprechenden Anwendung konfiguriert. Beispielsweise können den Properties Werte zugewiesen werden oder DeploymentArtifacts angehängt bzw. überschrieben werden. Abbildung 2.2 zeigt ein Beispiel für ein TopologyTemplate, das einen in Java implementierten Dienst beschreibt. Auf der virtuellen Maschine läuft dabei ein Betriebssystem, an das verschiedene DeploymentArtifacts für verschiedene Plattformen angehängt sind. Darauf ist ein Webserver installiert, dessen Eigenschaft `Port` im Modell konfiguriert wurde und das ein Betriebssystempaket als DeploymentArtifact enthält.

In dieser Abbildung wird, wie in der gesamten Arbeit, die grafische Notation „Vino4TOSCA“ [BBK+12] genutzt. Vino4TOSCA visualisiert NodeTemplates als abgerundete Rechtecke, die oben den Namen des NodeTemplates, darunter in Klammern den Namen des NodeTypes und im unteren Bereich alle weiteren Definitionen, wie zum Beispiel Properties und DeploymentArtifacts, darstellen.

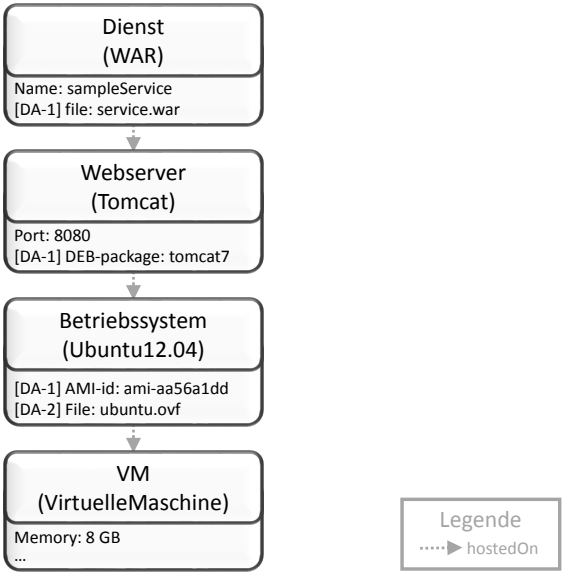

Abbildung 2.2: Beispiel TOSCA TopologyTemplate

2.4.3.2 Orchestrierung

TOSCA ermöglicht es, die Durchführung von Verwaltungsaufgaben zur Bereitstellung, Laufzeit (z.B. Datensicherung, Softwareaktualisierung) und Außerbetriebnahme zu automatisieren. Verwaltungsaufgaben werden hierbei durch die Kombination der von NodeTypes und RelationshipTypes angebotenen Operations beschrieben. Grundsätzlich sind in TOSCA, basierend auf der zuvor eingeführten Anwendungstopologie, zwei Arten der automatisierten Verarbeitung von Verwaltungsaufgaben vorgesehen [OAS13a, BBKL14a]: (i) Bei der *imperativen* Verarbeitung wird für jede Verwaltungsaufgabe ein ManagementPlan modelliert, der explizit die Abfolge der Operations und den Datenfluss enthält. Für die Beschreibung von ManagementPlans werden Geschäftsprozesse genutzt, für die bereits Laufzeitumgebungen zur automatisierten Ausführung existieren. Die Laufzeitumgebung muss nur die Pläne ausführen und sicherstellen, dass diese die Operations aufrufen können,

jedoch die Semantik der TOSCA-Anwendung nicht verstehen. Dies ermöglicht eine flexible Implementierung von Verwaltungsaufgaben, die jedoch mit dem Mehraufwand verbunden ist, Pläne zu modellieren. (ii) Bei der *deklarativen* Verarbeitung ist die Laufzeitumgebung für die Durchführung der Verwaltungsaufgaben verantwortlich und muss somit die entsprechende Logik implementieren. Dazu wird ausschließlich das TopologyTemplate interpretiert und die entsprechenden Operations aufgerufen. Die rein deklarative Verarbeitung ist daher besonders für die Bereitstellung und Außerbetriebnahme einer Anwendung geeignet und weniger für Verwaltungsaufgaben zur Laufzeit. Bei der deklarativen Verarbeitung wird die Flexibilität und Erweiterbarkeit durch die Annahmen, welche die Laufzeitumgebung trifft, reduziert. Gleichzeitig wird aber die Komplexität, insbesondere von Bereitstellung und Außerbetriebnahme, reduziert, weil kein ManagementPlan erstellt werden muss. Darüber hinaus sind verschiedene Mischformen denkbar, beispielsweise die Generierung eines ManagementPlans aus der Anwendungstopologie.

2.4.3.3 Paketierung und Laufzeitumgebung

Das „Cloud Service Archive“ (CSAR) paketiert die gesamte Anwendung, das heißt deren Anwendungstopologie, mitsamt allen Definitionen und Dateien, in einer ZIP-Datei. Außerdem, je nach gewählter Art für die automatisierte Verarbeitung von Verwaltungsaufgaben (vgl. Abschnitt 2.4.3.2), sind auch die Pläne enthalten. Dabei hat der Entwickler die Wahl, eine in sich geschlossene CSAR zu erstellen, die alle Daten enthält und damit entsprechend groß ist, oder große DeploymentArtifacts wie Images außerhalb der CSAR zum Beispiel über das Internet zur Verfügung zu stellen. Die CSAR wird von einer TOSCA-Laufzeitumgebung verarbeitet, welche die darin enthaltenen TOSCA-Definitionen interpretiert und umsetzt [BBLS12]. Dies beinhaltet beispielsweise die Bereitstellung der definierten Operations und Pläne.

2.4.3.4 Diskussion

TOSCA ist aktuell, im Vergleich zu den zuvor vorgestellten Ansätzen, eine der weitgehendsten Spezifikationen für die Modellierung von Anwendungstopologien. Außerdem werden die Automatisierung von Verwaltungsaufgaben und Portabilität zwischen verschiedenen Laufzeitumgebungen ermöglicht. TOSCA ist aufgrund seines offenen Konzeptes für NodeTypes und RelationshipTypes generisch und leicht erweiterbar.

Die Realisierung der vorliegenden Arbeit, welche in Kapitel 7 vorgestellt wird, verwendet TOSCA zur Beschreibung von Anwendungstopologien und CSARs als portables Austauschformat für Anwendungen. TOSCA ist jedoch nur eine Möglichkeit, die Forschungsbeiträge dieser Arbeit zu realisieren. Alternativ können andere Spezifikationen zur Beschreibung von Anwendungstopologien, die vergleichbare Konzepte wie TOSCA bieten, genutzt werden. Dies ist von Bedeutung, da TOSCA ein relativ neuer Standard ist, dessen Durchsetzung am Markt noch nicht abzusehen ist.

2.5 IT-Instanzmodelle

Zur Analyse, Adaption und Optimierung der IT ist es von zentraler Bedeutung, Informationen über den Aufbau der Anwendungen und deren Abhängigkeiten untereinander zu kennen [PX13] – insbesondere da Änderungen in der IT, beispielsweise aufgrund einer fehlenden detaillierten und globalen Sicht der gesamten IT, weitreichende negative Auswirkungen haben können [OGP03]. Daher ist ein Instanzmodell der gesamten IT nötig, das nicht nur einzelne Anwendungen betrachtet, sondern auch deren Abhängigkeiten untereinander. Dies ist wichtig, da Anwendungen üblicherweise nicht isoliert, sondern mit anderen internen und externen Anwendungen integriert sind. Darüber hinaus sind auch die technischen Details jeder einzelnen Anwendung erforderlich, also die eingesetzten Komponenten sowie deren Relationen und Konfiguration. Eine weitere Anforderung ist, dass es sich um ein aktuelles Instanzmodell mit aktuellen Laufzeitinformationen handelt, da sonst die Diskrepanz zwischen Realität und Modell zu Problemen

führen kann. Ein solches IT-Instanzmodell wird in der vorliegenden Arbeit als „Enterprise Topologie“ bezeichnet:

Definition 2.5 (Enterprise Topologie – informell)**.** Eine Enterprise Topologie ist ein technisch detailliertes Instanzmodell, welches die gesamte IT einer Organisation zu einem bestimmten Zeitpunkt repräsentiert. Diese enthält insbesondere auch die Relationen zwischen verschiedenen Anwendungen der Organisation.

Im Folgenden werden Enterprise Topologien zuerst von Anwendungstopologien (Abschnitt 2.5.1) und dem Enterprise Architecture Management (Abschnitt 2.5.2) abgegrenzt. Danach werden in Abschnitt 2.5.3 verwandte Arbeiten in verschiedenen Bereichen betrachtet und die gewonnenen Erkenntnisse in Abschnitt 2.5.4 diskutiert.

2.5.1 Abgrenzung von Anwendungstopologien

Enterprise Topologien unterscheiden sich von den in Abschnitt 2.4 betrachteten Anwendungstopologien in zwei Punkten: (i) Anstatt eines präskriptiven Modells [LL10] der Anwendungen wird ein deskriptives Instanzmodell abgebildet, also der Zustand der IT zu einem bestimmten Zeitpunkt. (ii) Anstatt einer einzelnen Anwendung liegt der Fokus auf der Repräsentation vieler Anwendungen und deren Relationen. Die anwendungsübergreifende detaillierte Darstellung führt dazu, dass mit sehr großen Enterprise Topologien zu rechnen ist. Gemeinsam haben Anwendungstopologien und Enterprise Topologien den Grad der technischen Details.

2.5.2 Abgrenzung von Enterprise Architecture Management

Das „Enterprise Architecture Management“ (EAM) hat das Ziel, ein gesamtheitliches Bild der Enterprise Architektur zu verwalten und deren Abgleich mit den Geschäftszielen sicherzustellen [WBMS10]. Um diese Ziele zu erreichen, deckt EAM die gesamte IT ab, enthält jedoch, aufgrund der strategischen Ausrichtung, nur wenige technische Details [BW07]. Beim

EAM werden, neben dem Modell des aktuellen Zustands, ein oder mehrere Zielmodelle definiert [BMR+10].

Beispiele für EAM-Methoden sind „TOGAF“ und „ArchiMate“: TOGAF (The Open Group Architecture Framework) [Jos09] ist eine EAM-Methode, die Architekturen modelliert, um die Implementierung und Evolution der Anwendung zu unterstützen. Dabei wird zwischen vier Typen von Architekturen unterschieden [Jos09]: (i) „Business“, (ii) „Daten“, (iii) „Anwendung“, und (iv) „Technologie“. Die Anwendungsarchitektur (iii) beschreibt dabei die abstrakten Funktionen und Schnittstellen, die zur Erfüllung der Geschäftsziele aus Architekturtyp (i) benötigt werden [The06], ohne deren Implementierung und technische Details zu betrachten. Die Technologiearchitektur (iv) definiert die einzelnen Dienste, welche die Funktion der Anwendung implementieren [The06]. Auch hier geht es nicht um technische Details, sondern um die Definition der Anforderungen für die Beschaffung oder Implementierung der Dienste. ArchiMate [The13] ist eine Modellierungssprache für Enterprise Architektur, die bewusst von den technischen Details abstrahiert [BGM+12]. Dabei ist es möglich, Teile der in TOGAF definierten Modelle mit ArchiMate zu modellieren [Cha12]. Trotz ähnlicher Begrifflichkeiten wird deutlich, dass die Modelle, die im EAM verwendet werden, kaum technische Details enthalten, auch weil sie andere Ziele verfolgen.

2.5.3 Verwandte Arbeiten zu IT-Instanzmodellen

Die „IT Domain-specific Modeling Language“ (ITML) [FHK+09] adressiert die Verbesserung der Relation zwischen IT und Geschäftszielen. Hierfür verwenden Frank et al. [FHK+09] verschiedene Perspektiven, die teilweise auch mit Instanz- und Monitoring-Informationen angereichert werden können. Die Modellierung der IT ist dabei beschränkt auf eine Reihe vordefinierter und abstrakter IT-Komponenten. Auch abstrakt, aber erweiterbar, ist der Ansatz von Machiraju et al. [MDW+99], der die „Unified Modeling Language“ (UML) [OMG11b] nutzt, um Anwendungsvorlagen und -instanzen zu repräsentieren. Die Anwendungsvorlagen definieren die möglichen Eigenschaften und Struktur der Instanzen. Ziel des auf die Anwendungsbe-

standteile Middleware und Anwendungsfunktionalität (vgl. Abschnitt 2.2) fokussierten Modells ist es, Varianten in der Konfiguration zu identifizieren und zu vergleichen. Auch auf UML basierend ist das „Common Information Model“ (CIM) [Dis13], ein Standard zur Beschreibung und zum Austausch von Verwaltungsinformationen über IT-Systeme, der Basis verschiedener anderer Standards [DFG+11] ist. CIM legt den Schwerpunkt bei der Definition und Nutzung auf die Repräsentation der Infrastruktur.[1] Scheibenberger und Pansa [SP08] erweitern CIM um die Möglichkeit, Relationen zwischen Attributen zu definieren, um beispielsweise abbilden zu können, dass zwei Anwendungen um die gleichen physikalischen Ressourcen konkurrieren.

Technische Informationen, wie die Konfiguration oder die installierte Software, werden auch in IT-Management-Lösungen verschiedener Anbieter verwaltet [LMPR07, JDBN+10]. Dabei werden „Configuration Items“ in einer zentralen Datenbank gehalten, der „Configuration-Management Database“ (CMDB). Die Instanzmodelle werden dabei in proprietären und, aufgrund der hohen Anzahl an Funktionen, komplexen Modellen abgelegt, die nur schwer von anderen Werkzeugen genutzt oder föderiert werden können [JDBN+10]. Den Relationen zwischen den Einträgen der CMDBs kommt nur eine untergeordnete Rolle zu und die darin gespeicherten Informationen entsprechen nicht immer dem aktuellen Stand der IT.

2.5.4 Diskussion und offene Forschungsfragen

Da sich viele der vorgestellten Instanzmodelle aus dem Bereich des „Systems Management“ entwickelt haben, ist dieser Fokus heute oft noch erkennbar, zum Beispiel beim Detailgrad bzw. bei der Granularität der verschiedenen Arten von Bestandteilen (vgl. Abschnitt 2.2) in den Modellen. Auch wenn Instanzmodelle heute alle Bestandteile von Anwendungen abbilden, sind die Anwendungsfunktionalität und die Geschäftsprozesse offensichtlich nicht der Fokus. Für manche Anwendungsfälle ist dies die passende Repräsentati-

[1] Das standardisierte Schema [Dis13] erlaubt es, basierend auf dem *Core*-Schema die folgenden Klassen zu benutzen: *Application*, *Application-J2eeAppServer*, *Database*, *Device*, *Event*, *Interop*, *IPsecPolicy*, *Metrics*, *Network*, *Physical*, *Policy*, *Security*, *Support*, *System* und *User*.

on der IT, bei der Migration stehen jedoch die Anwendungsfunktionalität und die Geschäftsprozesse im Mittelpunkt. Die darunterliegenden Komponenten, welche die Anwendungsfunktionalität und Geschäftsprozesse realisieren, treten dabei eher in den Hintergrund, weil sie bei der Migration verändert oder sogar ersetzt werden. Der Detailgrad von existierenden Ansätzen für Instanzmodelle reicht von der Abstraktion aller technischen Details bis zur ausschließlichen, aber detaillierten Beschreibung einzelner Komponenten oder Anwendungen. Falls die für Bereitstellung und Verwaltung verwendete Anwendungstopologie und das alle Anwendungen umfassende Instanzmodell in Relation miteinander gebracht werden sollen, ist es hilfreich, dieselbe Granularität zu wählen. Auch die Typisierung ist ein wichtiger Aspekt, so dass sich die Typsysteme der meisten Instanzmodelle und Anwendungstopologien erweitern lassen.

Zusammenfassend ist festzustellen, dass bisher kein Metamodell für eine (i) technisch detaillierte, (ii) alle Arten von Anwendungsbestandteilen umfassende, (iii) erweiterbare, (iv) für verschiedene Anwendungsfälle zugängliche, (v) anwendungsübergreifende und (vi) formale Enterprise Topologie existiert, die alle Komponenten aller Anwendungen und deren Relationen abbildet. Ohne dieses IT-Instanzmodell sind jedoch eine automatisierte Migration und das automatisierte Management laufender Anwendung im Allgemeinen nicht möglich [BBK+14b].

2.6 Erstellung von Enterprise Topologien

In diesem Abschnitt wird betrachtet, wie die in Definition 2.5 eingeführten Enterprise Topologien erstellt werden können. Diese Instanzmodelle existieren normalerweise nicht, da die IT-Komponenten manuell verwaltet werden, sich regelmäßig weiterentwickeln und keine Dokumentation geführt wird. Meistens sind nicht einmal die Abhängigkeiten zwischen verschiedenen Komponenten bekannt [KBK00]. Auch Hersteller von Anwendungen liefern nur textuelle und abstrakte Anforderungen und keine Anwendungstopologien für ihre Anwendungen [Ens99].

Grundsätzlich kann eine Enterprise Topologie entweder durch manuelle Modellierung oder automatisierte Generierung erstellt werden. Die manuelle Erstellung von Enterprise Topologien ist mit hohen Kosten verbunden, da sie zeitaufwändig und fehleranfällig ist [Ens99, MDW+99, FAB+10]. Insbesondere eine regelmäßige Aktualisierung ist deshalb nicht möglich [MDW+99]. Dies führt dazu, dass bei der manuellen Erstellung Enterprise Topologien mit einem hohen Abstraktionsgrad erstellt werden [WBMS10, BLNS12] die meist nicht konsistent gehalten werden [MDW+99]. Brown et al. [BKK01] kommen zu dem Schluss, dass die manuelle Erstellung durch Experten nur bei kleinen und einfachen Anwendungen möglich ist und somit nicht für Enterprise Topologien anwendbar ist.

Aufgrund der zuvor aufgeführten Schwierigkeiten bei der manuellen Modellierung von Enterprise Topologien beschäftigt sich die vorliegende Arbeit ausschließlich mit der automatisierten Erstellung von Enterprise Topologien. Aufgrund der großen Anzahl von verwandten Arbeiten wurden diese gruppiert und in verschiedene Abschnitte aufgeteilt.

2.6.1 Automatisierte Enterprise Architektur Dokumentation

Der Forschungsbereich der automatisierten Enterprise Architektur Dokumentation hat das Ziel, verschiedene Perspektiven der Enterprise Architektur (EA) automatisiert zu erstellen. Ein Beispiel dafür ist die automatisierte Dokumentation der Technologiesicht einer EA aus verschiedenen Informationsquellen, beispielsweise durch Extraktion der Informationen aus einem Enterprise Service Bus [BGM+12] oder einem Scanner für sicherheitskritische Schwachstellen in Unternehmen [BHS+12]. Ritter [Rit12] extrahiert die Abhängigkeiten zwischen und innerhalb von Firmen auf der Geschäftsebene, basierend auf den technischen Abhängigkeiten, die in der IT zu finden sind. Farwick et al. [FAB+10] übertragen Informationen von Infrastrukturkomponenten in die EA, ohne dabei neue Relationen zu finden. Dazu wird die Komponente, sowohl in der EA als auch dem System, welches die Komponente verwaltet, mit einer eindeutigen ID annotiert. Mithilfe dieser ID werden die Laufzeitinformationen regelmäßig in die EA propa-

giert. Hauder et al. [HMR12] beschreiben die offenen Forschungsfragen in diesem Bereich und Farwick et al. [FAB+11a] die nötigen organisatorischen Prozesse, um eine EA zu erstellen und zu pflegen.

Enterprise Architekturen haben einen deutlich höheren Abstraktionsgrad als Enterprise Topologien (vgl. Abschnitt 2.5.2), jedoch sind die auftretenden Herausforderungen vergleichbar und deshalb relevant für die Erstellung von Enterprise Topologien. Die vorliegende Arbeit positioniert Enterprise Architekturen als Modell, um strategische und langfristige Entscheidungen über die Weiterentwicklung der IT zu treffen und deren Relation zu den Geschäftszielen sicherzustellen. Diese Entscheidungen werden dann, auf technischer Ebene, mithilfe von Enterprise Topologien realisiert. Dem folgend wurde in [BLNS12] ein Ansatz zur automatisierten Abstraktion einer Enterprise Topologie in eine Enterprise Architektur vorgestellt, der erlaubt, abstrakte EAs aus technisch detaillierten Enterprise Topologien zu erstellen. Ein ähnlicher Ansatz wird von Fischer et al. [FAW07] verfolgt, die Enterprise Architekturen aus verschiedenen Architekturmodellen (z.B. Software-, Integrations- und Prozessarchitekturen) zusammensetzen.

2.6.2 Software Architecture Reconstruction

Wo keine Architekturdokumentation (mehr) vorhanden ist, hilft „Software Architecture Reconstruction“ (SAR) die Architektur einer existierenden Software zu dokumentieren, zu verstehen und zu verbessern [Gar00, PDP+07, DP09]. Pollet et al. [PDP+07] unterscheiden dabei zwischen (i) *bottom-up*, das heißt, der Code der Anwendung wird abstrahiert, (ii) *top-down*, das heißt, es wird versucht, noch vorhandene Anforderungen oder den dokumentierten Architekturstil auf den Code abzubilden, und (iii) hybriden Ansätzen. Mendonça und Kramer [MK01] betrachten insbesondere verteilte Systeme und erstellen verschiedene Sichten auf diese. SAR beschäftigt sich mit der Architektur und nicht mit Instanzmodellen, weshalb auch normalerweise keine Instanz der Anwendung untersucht wird, sondern deren Code und Dokumentation.

2.6.3 Identifikation von Abhängigkeiten zwischen Diensten

Dieser Abschnitt betrachtet Arbeiten, welche die Abhängigkeiten zwischen bekannten Diensten auf verschiedene Weisen identifizieren.

Basu et al. [BCD08] identifizieren Relationen zwischen Diensten aufgrund von kausalen Abhängigkeiten in den Dienstaufrufen. Wenn Dienst A aufgerufen wurde und immer kurz danach Dienst B aufgerufen wird, kann daraus geschlossen werden, dass zwischen beiden eine Abhängigkeit besteht. Ensel [Ens99] berechnet die Abhängigkeiten zwischen einer Menge von gegebenen Diensten mithilfe von neuronalen Netzwerken. Dabei wird die aufgezeichnete Prozessor- und Netzwerkauslastung der entsprechenden Maschinen paarweise betrachtet, um Muster zu identifizieren, die auf eine Abhängigkeit zwischen den von diesen Maschinen betriebenen Diensten hindeuten. Im Gegensatz dazu identifizieren Brown et al. [BKK01] Abhängigkeiten, indem sie gezielt charakteristische Workloads ausführen und durch Analyse der Monitoring-Daten aller Komponenten analysieren, ob diese für die Verarbeitung dieses Workloads genutzt wurden. Dies wird für jede Komponente wiederholt. Die Herausforderung ist, dass alle Komponenten überwacht werden müssen und für jede Komponente bekannt sein muss, wie man auf dieser Workload generiert. Deshalb ist die Messung auch nur in einer Testumgebung möglich, da nebenläufige Workloads die Ergebnisse verfälschen können. Durch statistische Korrelation wird anschließend das Gewicht der Abhängigkeit der Komponenten bestimmt. Bei der Analyse von Performance-Problemen (engl. *root cause analysis* [Ens99]) ist dieser Ansatz hilfreich, was auch das Ziel der Autoren war. Für die Erstellung von Enterprise Topologien sind die zuvor vorgestellten Ansätze allerdings nicht geeignet, weil sie nicht in Produktionsumgebungen eingesetzt werden können. Außerdem wird mit der Auslastung nur eine Art der Abhängigkeit von Diensten betrachtet, die aber nicht zwangsläufig alle für den Betrieb der Anwendung nötigen Abhängigkeiten bestimmt: Beispielsweise funktioniert eine Anwendung ohne Benutzerverwaltung oder DNS-Server möglicherweise nicht, obwohl in der Auslastung keine Korrelation erkennbar ist, weil einzelne Anfragen an solche Systeme nur wenige Ressourcen konsumieren oder zwischengespeichert werden (engl. *caching*).

Keller et al. [KK01, EK02] finden Abhängigkeiten zwischen Diensten durch das kontrollierte Auslösen von Fehlern in den zu untersuchenden Systemen und Informationen aus Konfigurationsdatenbanken. Dieser Ansatz bestimmt die Existenz einer Kommunikation von Diensten untereinander, aber nicht die technischen Abhängigkeiten zwischen den Komponenten, welche diese Dienste bereitstellen.

Die Identifikation von Abhängigkeiten auf Netzwerkebene ist sinnvoll, da die gesamte Kommunikation von Komponenten und Diensten, die auf verschiedenen Maschinen betrieben werden, über das Netzwerk läuft. Jedoch können nur externe Abhängigkeiten auf Netzwerkebene erkannt werden, da die Kommunikation der Komponenten und Dienste innerhalb einer (virtuellen) Maschine nicht im Netzwerk sichtbar ist und somit auch nicht die interne Abhängigkeiten bzw. Struktur der Anwendung. Außerdem spielt der Zeitpunkt der Ausführung eine entscheidende Rolle, da manche Verbindungen nur selten hergestellt und diese Informationen dann nicht gefunden werden können [LaC07]. Die Herausforderung bei diesen Ansätzen ist es, aus der Menge der Informationen die richtigen Schlüsse für die Erstellung der Enterprise Topologie zu ziehen. Arbeiten in diesem Bereich zeigen, beispielsweise basierend auf mithilfe von „NetFlow“ [Cla04] aufgezeichneten Daten, wie sich Relationen auch über mehrere Maschinen hinweg extrahieren lassen [KGE06, CKG+08]. Chen et al. [CZMB08] liefern vergleichbare Abhängigkeitsinformationen durch Nutzung eines „package sniffer“ auf dem Netzwerkrouter. Allerdings wird der Inhalt der Netzwerkpakete nicht analysiert. Es werden somit nur die Existenz einer Relation und das verwendete Protokoll ermittelt, nicht aber deren Semantik.

Um im Netzwerk vorhandene Anwendungen und deren Interaktion zu bestimmen, analysiert der „EMC Smarts Application Discovery Manager“ [BGL07] auch den Inhalt einer Untermenge der Netzwerkpakete. Dabei wird versucht, Anwendungen mithilfe einer Liste bekannter Anwendungen und typischer Interaktionen zu identifizieren. Im Gegensatz dazu erstellen Black et al. [BDF04] eine Netzwerktopologie basierend auf der Route von Testpaketen durch das Netzwerk, ohne dabei Netzwerkkomponenten zu benötigen, die NetFlow oder ähnliche Technologien unterstützen. Statt-

dessen muss dafür auf einem Teil der Server im Netzwerk eine Software installiert werden, welche die Testpakete verarbeitet, was aber die Nutzung des Ansatzes in der Praxis erschwert. Durch die Kombination verschiedener Informationen und Logs aus dem Netzwerk erstellen Gantenbein und Deri [GD02] automatisiert eine Bestandsaufnahme (engl. *asset discovery*) der Netzwerkgeräte, um deren Nutzung und Position verfolgen zu können.

2.6.4 Erstellung von Instanzmodellen großer Teile der IT

Dieser Abschnitt stellt Ansätze und Produkte vor, welche die Erstellung von Instanzmodellen von großen Teilen der IT ermöglichen. Solche Lösungen findet man, mit den in diesem Abschnitt diskutierten Einschränkungen, in verschiedenen Angeboten von großen Anbietern für IT-Management, aber auch verwandten Arbeiten in der Wissenschaft.

Der „IBM Tivoli Application Dependency Discovery Manager“ (TADDM) [IBM14] erstellt Enterprise Topologien und bietet darauf basierend verschiedene Funktionalitäten zur Analyse und zum Management an. Dabei wird keine installierte Software auf den Komponenten der Enterprise Topologie benötigt, sondern die Informationen werden *von außen* extrahiert. Dies reduziert die technischen und organisatorischen Anforderungen, weil diese Software nicht installiert, konfiguriert oder verwaltet werden muss. Neben der engen Integration in die Herstellerplattform kann TADDM nur eine begrenzte Anzahl bekannter Softwarepakete erkennen und benötigt zur Identifikation von Geschäftsanwendungen eine Vorlage welche deren Struktur und Komponenten beschreibt [MKLT13]. Auf Basis der veröffentlichten Informationen vergleichbar damit ist „HP Universal Discovery“ [HP12] (früher „HP Discovery and Dependency Mapping“), das eine Menge von *discovery patterns* kombiniert, die jeweils einen kleinen Teil der Enterprise Topologie erstellen. Basierend auf diesen Daten kann der Nutzer zum Beispiel Veränderungen an der Konfiguration der Komponenten verfolgen oder die Einhaltung von Vorgaben prüfen.

Bei den Arbeiten in der Wissenschaft[1] nutzt „DepMap“ [Mat10] Ereignisse im Betriebssystem, um Abhängigkeiten einer Anwendungsinstanz von externen Komponenten über das Netzwerk zu identifizieren und zu überwachen. Ziel ist die Optimierung der Ressourcennutzung und Performanz der Anwendung. Eine Enterprise Topologie, die mehrere Betriebssysteme umfasst, kann nicht erstellt werden. Machiraju et al. [MDW+99] beschreiben eine Lösung, welche die Konfiguration von Anwendungsinstanzen extrahiert und Änderungen oder Variationen zwischen Instanzen der Anwendung erkennt. Dabei muss für jede Anwendung eine Vorlage definiert werden, welche die zu überprüfenden Eigenschaften der Anwendung beschreibt und angibt, wie diese ermittelt werden. Im Gegensatz dazu stellen Meng et al. [MZY+13] einen generischen Ansatz vor, der relevante Konfigurationsdateien automatisiert extrahiert. Dabei werden mithilfe der Checksumme aller Dateien auf dem jeweiligen System Unterschiede zum vordefinierten *Benchmark-Image* festgestellt und mithilfe einer Klassifikation die Konfigurationsdateien extrahiert, ohne anwendungsspezifisches Wissen zu benötigen. Die Struktur oder Abhängigkeiten werden jedoch nicht bestimmt.

Die vorgestellten Ansätze basieren überwiegend auf einem Katalog von bekannten Komponenten mit Signaturen und sind auf eine Menge an Anwendungen (oder Images) beschränkt. Die Lösungen der großen Anbieter für IT-Management fokussieren eher die Bestandsaufnahme von Komponenten, deren Konfiguration, Verwaltung von Lizenzen und Monitoring, aber nur sekundär deren detaillierte technische Relationen, wie sie für Bereitstellung oder Migration benötigt werden. Dabei sind diese Lösungen oft komplex, schwer installierbar, benötigen die Definition von Vorlagen, um Anwendungen zu erkennen, und sind, auch durch die Nutzung von proprietären Formaten [LaC07], stark in andere Angebote des jeweiligen Anbieters integriert [JDMV08].

[1] Die Autoren der verwandten Arbeiten aus der Wissenschaft sind bzw. waren teilweise bei den oben genannten Firmen angestellt, es wird aber keine Verbindung zum jeweiligen Produkt hergestellt, wodurch nicht bekannt ist, ob diese Ansätze in den Produkten umgesetzt wurden.

2.6.5 Komponentenspezifische Extraktion von Laufzeitinformationen

Dieser Abschnitt stellt Lösungen vor, die Laufzeitinformationen von einzelnen Komponententypen extrahieren. Im Betriebssystem integrierte oder darauf installierte Werkzeuge erlauben es beispielsweise, detaillierte technische Informationen über Hardware, Betriebssystem und installierte Software abzufragen. Ein Beispiel für ein in das Betriebssystem integriertes Werkzeug ist das „proc“-Dateisystem von Linux [BBN+09], das den Zugriff auf eine Vielzahl von Informationen über das System ermöglicht. Plattformübergreifende Werkzeuge aus dem DevOps-Bereich, wie „Opscode Ohai“ [Che14b] oder „Puppet Labs Facter“ [Pup09], erlauben es, für die automatisierte Bereitstellung nötige Informationen wie die Hardwareausstattung, Version und Konfiguration des Betriebssystems zu ermitteln.

Informationen über Geräte und Software, die über das Netzwerk erreichbar sind, lassen sich aufgrund ihres charakteristischen *Fingerabdrucks* ermitteln. Der Fingerabdruck wird basierend auf den Headern verschiedener Protokolle, der Sortierung dieser Header, installierter Erweiterungen, dem Verhalten im Fehlerfall usw. ermittelt. Mit dieser Methode können zum Beispiel der Typ und die Version von Webservern [Net05] oder Betriebssystemen [Lyo14] herausgefunden werden.

Eine komponentenspezifische Lösung für den Bereich der Datenspeicherung ist beispielsweise „Galapagos“ [MDJV08, JPRD10], das ein Instanzmodell mit den Abhängigkeiten zwischen Anwendungen und den von diesen genutzten Daten (Dateien, Datenbanken, Festplatten etc.) erstellt. Die auf den verschiedenen Servern gesammelten Daten werden dann analysiert und zusammengeführt. In einer weiterführenden Arbeit untersuchen Joukov et al. [JTO+11] JavaEE-Anwendungen und stellen ein Werkzeug vor, das in deren Bytecode Referenzen auf externe Ressourcen findet, wie zum Beispiel Datenbanken und Dateien. Darüber hinaus können diese Referenzen angepasst werden, beispielsweise um eine Datei zukünftig von einem anderen Ort zu laden.

Alle in diesem Abschnitt vorgestellten Ansätze haben gemeinsam, dass sie Teile oder bestimmte Aspekte einer Enterprise Topologie extrahieren können. Aufgrund der Spezialisierung und des Expertenwissens, das in diese Ansätze geflossen ist, sind die Ergebnisse technisch detailliert und vollständig.

2.6.6 Diskussion und offene Forschungsfragen

Die Übersicht der Ansätze zur (teil)automatisierten Erstellung von Enterprise Topologien zeigt, dass es bisher keine Lösung gibt, die eine komplette Enterprise Topologie, wie im Kontext dieser Arbeit definiert, erstellen kann. Existierende Lösungen sind entweder auf bestimme Komponenten beschränkt (siehe Abschnitt 2.6.5), bestimmen die Struktur oder Abhängigkeiten nur auf Ebene der Dienste oder des Netzwerks (siehe Abschnitt 2.6.3), erkennen lediglich bestimmte, vordefinierte Anwendungen (siehe Abschnitt 2.6.4) oder sind zu abstrakt und nicht auf technische Details fokussiert (siehe Abschnitt 2.6.1 und Abschnitt 2.6.2). Der Abschnitt zeigt aber auch, dass es eine Vielzahl von bewährten Lösungen gibt, die einen Teil der Enterprise Topologie abdecken, und dass somit eine Integration verschiedener Informationsquellen und Werkzeuge sinnvoll ist.

In Bezug auf die Arten von Anwendungsbestandteilen (Abschnitt 2.2) ist festzustellen, dass Geschäftsprozesse und deren Dienstaufrufe durch Enterprise Service Busse nicht von den existierenden Arbeiten adressiert werden [LaC07]. Da Enterprise Service Busse das zentrale Element jeder Service-orientierten Architektur sind (vgl. Abschnitt 2.1) und auch die Orchestrierung von Diensten durch Geschäftsprozesse oft eingesetzt wird, führen Lösungen, welche diese nicht betrachten, zu unvollständigen Enterprise Topologien. Insbesondere beim Anwendungsfall der Migration sind die Anwendungsfunktionalität und Geschäftsprozesse jedoch zentral. Alle Komponenten darunter sind nur für den Betrieb derselben verantwortlich.

2.7 Migration von Anwendungen

Die Migration einer existierenden Anwendung hat das Ziel, diese in einer anderen Zielumgebung zu betreiben. Im Kontext der vorliegenden Arbeit wird als Zielumgebung exemplarisch die Cloud betrachtet. In Anlehnung an Ackermann et al. [AGW05], Sneed et al. [SWH10] und Jamshidi et al. [JAP13] wird eine Anwendungsmigration wie folgt definiert:

Definition 2.6 (Anwendungsmigration – informell)**.** Eine Anwendungsmigration bezeichnet die Überführung einer Anwendung aus einer Ursprungsumgebung in eine Zielumgebung. Dabei werden alle nötigen Anpassungen an der Anwendung durchgeführt, deren Anwendungsfunktionalität und Geschäftslogik bleiben dabei aber unverändert.

Hervorzuheben ist, dass bei der Migration ausschließlich die Änderungen vorgenommen werden, welche für die Ausführung in der Zielumgebung nötig sind. Weitergehende Optimierungen, beispielsweise hinsichtlich der Architektur, Skalierbarkeit oder Mehrmandantenfähigkeit der Anwendung, sind nicht Teil der eigentlichen Migration, sondern der Weiterentwicklung der Anwendung [AGW05, SWH10]. Dies ist auch ein Grund dafür, dass die in die Cloud migrierten Anwendungen selten native Cloud-Anwendungen sind (vgl. Abschnitt 2.3.2).

Eine systematische Literaturanalyse von Jamshidi et al. [JAP13] kam zu dem Ergebnis, dass die Forschung im Bereich der Migration von Anwendungen in die Cloud noch am Anfang steht und insbesondere Werkzeugunterstützung für die Automatisierung von Migrationsaufgaben fehlt. Dabei teilen Jamshidi et al. die relevanten Arbeiten in die in Abbildung 2.3 dargestellten Klassen (i) Querschnittsthemen, (ii) Planung, (iii) Durchführung und (iv) Evaluation ein [JAP13]. *Querschnittsthemen* beinhalten Aspekte, welche die gesamte Migration übergreifen, von der Organisation, über die Sicherheitsanalyse, bis hin zur Mitarbeiterqualifikation. In der *Planungsphase* werden beispielsweise Anbieter und Dienste für die Zielumgebung ausgewählt, die Umsetzbarkeit der Migration bewertet und Anforderungen gesammelt. Während der *Durchführung* wird die Architektur ermittelt,

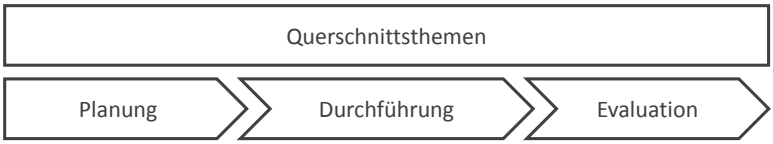

Abbildung 2.3: Klassifikation verwandter Arbeiten zur Migration von Anwendungen (angelehnt an [JAP13])

werden die Anwendungsdaten extrahiert sowie der Code und andere Komponenten angepasst. Die *Evaluation* führt die Bereitstellung der Anwendung durch, um die Migration zu validieren, und testet die migrierte Anwendung. Darüber hinaus wurde in jeder Kategorie die Automatisierung der Aufgaben untersucht, wobei sich zeigte, dass in der Kategorie Durchführung die wenigsten Arbeiten zur Automatisierung existieren. Diese Lücke, die technische Umsetzung und Automatisierung der Durchführung einer Anwendungsmigration, wird durch die vorliegende Arbeit adressiert.

Daher legt dieser Abschnitt den Schwerpunkt auf verwandte Arbeiten im Bereich der Automatisierung der Migrationsdurchführung, die in Abschnitt 2.7.3 vorgestellt werden. Die Grundlage dafür ist die Einführung verschiedener Migrationsstrategien in Abschnitt 2.7.1 und die Betrachtung nicht-automatisierter Vorgehensmodelle zur Migration von Anwendungen in Abschnitt 2.7.2.

2.7.1 Migrationsstrategien

Migrationsstrategien beschreiben, hinsichtlich der Fragen, welche Komponenten migriert werden und in welcher Reihenfolge, grundsätzliche Arten der Herangehensweise an eine Anwendungsmigration. Dabei wird in der Literatur allgemein zwischen drei Klassen unterschieden:

Migrationsstrategie 1: Migration der kompletten Anwendung. Die Anwendung wird komplett und auf einmal migriert, wobei diese vorübergehend nicht verfügbar ist. Diese Strategie wird in der Literatur als „Cold Turkey“ [BS95], „Big Bang“ [BLW+97], „Forklift“ [Pia09, FLR+13] oder „Punktübergabe“ [SWH10] bezeichnet.

Migrationsstrategie 2: Migration von Teilen der Anwendung. Es werden Teile einer großen Anwendung in mehreren Durchläufen migriert, bis die gesamte Anwendung migriert wurde. Nach jedem Durchlauf ist die Anwendung wieder lauffähig. Welche Komponenten der Anwendung wann migriert werden, wird in verschiedenen Ausprägungen dieser Migrationsstrategien unterschieden, beispielsweise ob zuerst die Daten und dann die anderen Teile der Anwendung migriert werden sollen oder umgekehrt [BS95, BLW+97]. Diese Strategie wird in der Literatur als „Fastforward Migration“ bzw. „Reverse Migration“ [BS95], „Database first“ bzw. „Database last“ [BLW+97], „Phased“ [Pia09] oder „Inkrementelle Paketumstellung“ [SWH10] bezeichnet.

Migrationsstrategie 3: Migration ohne Unterbrechung der Verfügbarkeit. Die Migration der Anwendung, ohne dass die Verfügbarkeit für den Endbenutzer unterbrochen oder eingeschränkt wird, verlangt eine Reihe von zusätzlichen Maßnahmen und macht die Migration deutlich komplexer [Pia09]. Diese Strategie ist genau genommen orthogonal zu den ersten beiden, da sie beschreibt, wie eine Migration durchgeführt wird, unabhängig davon, ob die Anwendung in einem (Migrationsstrategie 1) oder in mehreren Teilen (Migrationsstrategie 2) migriert wird. Trotzdem wird sie in der Literatur separat aufgeführt, beispielsweise unter dem Namen „Zero-downtime“ [Pia09] oder „Stateless Component Swapping“ [FLR+13].

Welche der Migrationsstrategien letztendlich verwendet wird, hängt von spezifischen Gesichtspunkten der Organisation und Anwendung ab. Die in der vorliegenden Arbeit vorgestellte Migrationsmethode erlaubt die Umsetzung aller genannten Migrationsstrategien.

2.7.2 Vorgehensmodelle zur Anwendungsmigration

Die Durchführung einer Anwendungsmigration verlangt eine Reihe vorhergehender und nachfolgender Aktivitäten, die nötig sind, um die Migration organisatorisch umzusetzen. Dieser Abschnitt betrachtet verschiedene Vorgehensmodelle, welche die Migration von Anwendungen beschreiben.

Anbieter von Cloud-Diensten, beispielsweise Amazon [Var10] und Microsoft [BHJ+13], haben Methoden, Prozesse und Anwendungsfälle veröffent-

licht, um Anwendungen in ihre Cloud zu migrieren. Amazon empfiehlt eine Migration in sechs Phasen: (i) *Bewertung* hinsichtlich Kosten, Architektur und Sicherheit, (ii) *Proof of Concept* mit einer Pilot-Anwendung, (iii) *Migration der Daten*, (iv) *Migration der Anwendungen*, (v) *Cloud Funktionalitäten nutzen*, z.B. Automatisierung oder Elastizität, und (vi) *Optimieren*, z.B. der Auslastung durch Monitoring und Skalierung. Eine Auswahl des Anbieters und die Verteilung der Anwendung auf verschiedene Anbieter sind nicht vorgesehen, da dieses Vorgehensmodell nur die Migration zu Amazon betrachtet. Wie auch in Definition 2.6 beschrieben, sieht Amazon vor, die Anwendung erst in die Cloud zu migrieren und diese danach, in Phase 5 und 6, hinsichtlich ihrer Nutzung der Cloud zu optimieren. Die Migration der Daten (Phase 3) vor der Anwendung (Phase 4) birgt das Risiko, dass sich die Daten während der Migration verändern oder die Anwendung für die gesamte Dauer der Migration nicht verfügbar ist. Deshalb erscheint es vorteilhaft, zuerst die Anwendung zu migrieren und zu testen, bevor die Anwendungsdaten aus der Ursprungsumgebung migriert werden.

Microsoft empfiehlt, beispielsweise für die Migration datenintensiver Anwendungen [CTG+12], eine Migration in fünf Phasen: (i) *Analyse* hinsichtlich der Anwendungsarchitektur und deren Abbildung auf die Windows Azure Cloud, (ii) *Migration der Anwendungen*, (iii) *Migration der Daten*, (iv) *Optimierung und Test* sowie (v) *Betrieb und Verwaltung*. Wie auch bei Amazon ist nur die Abbildung von Komponenten auf Dienste von Microsoft Azure vorgesehen (Phase 1), aber keine Aufteilung der Anwendung auf verschiedene Anbieter. Im Gegensatz zu Amazon empfiehlt Microsoft, die Daten nach der Migration der Anwendung zu migrieren (Phase 3). Auch dieses Vorgehensmodell migriert erst die Anwendung in die Cloud und optimiert diese danach hinsichtlich ihrer Nutzung der Cloud-Eigenschaften.

Daneben wurden Vorgehensmodelle entwickelt, die unabhängig von einem einzelnen Anbieter sind: Ackermann et al. [AGW05] stellen, basierend auf verschiedenen Migrationsstrategien, einen Prozess für die Anwendungsmigration aus Sicht der Softwareentwicklung vor und haben diesen in den „Rational Unified Process“ [Kru04] eingebunden. Der daraus resultierende generische Referenzprozess für die Anwendungsmigration besteht aus

sieben Phasen: (i) *Anforderungs-Analyse*, (ii) *Legacy-Analyse* durch Aufbereitung des Designs der Anwendung, (iii) *Entwicklung des Ziel-Designs*, gegebenenfalls mit Zwischenschritten, (iv) *Strategieauswahl*, (v) *Implementierung*, (vi) *Qualitätssicherung* durch Tests, insbesondere Regressionstests, da sich die Funktionalität nicht ändern darf, und zuletzt der (vii) *Übergabe* (engl. *cut-over*), während der die konkrete Umstellung von der alten auf die neue Version der Software stattfindet. Bei der Strategieauswahl (iv) wird für jeden Anwendungsteil entschieden, ob eine Konversion, Kapselung oder Neuentwicklung die beste Lösung ist und ob die Anwendung in Teilen oder komplett migriert werden soll. Über die Software hinaus geht der „CloudMIG"-Ansatz [FH11a], der in Abschnitt 2.7.3 vorgestellt wird. Dieser beinhaltet einen Schritt zur Auswahl des Cloud-Anbieters und zur Ermittlung der nötigen Änderungen an der zu migrierenden Anwendung. Die eigentliche Durchführung der Migration und deren Automatisierung werden darin nicht näher betrachtet. Mohan [Moh11] stellt einen etwas technischeren, iterativen Prozess von Infosys Research vor, welcher die sieben Phasen (i) *Bewertung der Migration*, (ii) *Bestimmung der Abhängigkeiten* (Laufzeitumgebung, Lizenzen, Abhängigkeiten von anderen Anwendungen), (iii) *Aufteilung* der Anwendung zwischen der Cloud und dem lokalen Rechenzentrum, (iv) *Architektur anpassen* und gegebenenfalls nötige Implementierungsarbeiten, (v) *zusätzliche Cloud-Funktionalitäten nutzen*, (vi) *testen* und (vii) *optimieren* umfasst. In Ergänzung zu den genannten Ansätzen stellen Chauhan und Babar [CB12] einen manuellen Prozess vor, der die Aktivitäten vor der eigentlichen Durchführung der Migration beinhaltet, zum Beispiel die Auswahl der Anwendungen, welche migriert werden sollen.

Mit den organisatorischen (z.B. beteiligte Rollen), rechtlichen (z.B. Sicherheitskonzept) und wirtschaftlichen Fragestellungen einer Migration (z.B. Bewertung und Auswahl des Cloud-Anbieters) setzt sich das Vorgehensmodell von Vossen et al. [VHH12] auseinander. Die fünf Phasen sind: (i) *Vorbereitung und Planung*, (ii) *Auswahl eines Cloud-Anbieters*, (iii) *Vertragsgestaltung und Detailplanung*, (iv) *Umsetzung und Migration* sowie (v) *Betrieb*. Die eigentliche Durchführung der Migration wird nicht näher betrachtet, genauso wie die Optimierung der Anwendung für die Cloud. „Cloudstep" [BCX+12]

ist ein Entscheidungsprozess, der den Entitäten einer Anwendungsmigration eines von mehreren vordefinierten Profilen zuweist und Konflikte zwischen den Profilen findet, die vor einer Migration ausgeräumt werden müssen. Die wichtigsten Entitäten einer Anwendungsmigration sind dabei nach Beserra et al. [BCX+12] die Organisation, welche die Migration durchführen will, die zu migrierende Anwendung und die in Frage kommenden Cloud-Anbieter.

Im Bereich der Musterforschung [Wil12, FLR+14] wurden wiederkehrende und erprobte Vorgehensweisen bei der Anwendungsmigration als Muster (engl. *patterns*) abstrahiert: Fehling et al. [FLR+13] beschreiben Muster zur Verwaltung serviceorientierter Anwendungen, beispielsweise die Migration oder den Austausch einzelner Komponenten mit und ohne Unterbrechung von deren Verfügbarkeit. Pahl und Xiong [PX13] abstrahieren, basierend auf Fallstudien von Anwendungsmigrationen auf Plattformdienste, Prozessfragmente für verschiedene an der Migration beteiligte Rollen.

Alle vorgestellten Ansätze bewegen sich auf einem verhältnismäßig hohen Abstraktionsgrad und bieten kaum Werkzeugunterstützung bei der Durchführung der Migration, insbesondere nicht für deren Automatisierung. Daneben wird angenommen, dass die technischen Details der Anwendung, die migriert werden soll, schon bekannt sind. Dies ist aber, wie in Abschnitt 2.5 diskutiert, in der Praxis nicht der Fall.

2.7.3 Automatisierung der Migrationsdurchführung

Im Bereich der Anwendungsmigration in die Cloud basieren die meisten Arbeiten auf der Migration von Images bzw. virtuellen Maschinen [ABLS13]. Grund dafür ist der hohe Grad an Standardisierung und Portabilität von Images, was die Migration vereinfacht. Wie in Abschnitt 2.2 diskutiert, sind Anwendungen jedoch keine monolithischen Blöcke, sondern bestehen aus verschiedenen Komponenten, was in diesen Ansätzen nur teilweise berücksichtigt wird. Im Folgenden werden deshalb zur Migration von virtuellen Maschinen nur einige ausgewählte Ansätze vorstellt, die auch die Inhalte des migrierten Images betrachten.

Die Veränderung der Umgebung, in welcher die virtuelle Maschine läuft, hat zur Folge, dass Anpassungen im Image nötig sind. Sethi et al. [SSS+11] suchen nach dem Start der virtuellen Maschine in der Zielumgebung nach installierten Anwendungen und deren Konfiguration. Anwendungsspezifische Plugins korrigieren die sich durch die Migration in die Zielumgebung verändernden Netzwerkeinstellungen und fragen weitere Konfigurationsparameter, die geändert werden müssen, beim Nutzer ab. Andere Ansätze beschäftigen sich mit der Migration von Anwendungen, die über mehrere virtuelle Maschinen – und somit auch Images – verteilt sind: Pfitzmann und Joukov [PJ11] bilden die zu migrierenden Images auf vordefinierte und für die Zielumgebung optimierte Images ab, anstatt die ursprünglichen Images weiterzuverwenden. Diese Vereinheitlichung reduziert die Anzahl der Images und senkt somit den Verwaltungsaufwand. Gunka et al. [GSK13] schlagen einen evolutionären Ansatz vor, der versucht die Nutzung der Cloud-Eigenschaften durch die Anwendungen in den migrierten virtuellen Maschinen zu verbessern. Dabei werden zuerst alle virtuellen Maschinen migriert, dann entsprechende Loadbalancing-Mechanismen integriert und zuletzt, wo sinnvoll, Teile der Anwendung auf Plattformdienste umgestellt. Alle genannten Schritte zur Anpassung der Anwendung sind jedoch manuell durchzuführen. Die Migration einer großen Anzahl an Images wird von Al-Kiswany et al. [AKSSR11] optimiert, indem die Unterschiede zwischen den Images bestimmt und nur diese übertragen werden.

Anstatt ein komplettes Image zu migrieren, extrahieren Liu et al. [LLM+08] die installierten Unix-Pakete und deren Konfiguration in ein Modell und stellen daraus die gleiche Umgebung in der Zielumgebung wieder her. Auf dem Modell lassen sich zum Beispiel auch die Softwareversionen der Pakete vereinheitlichen. Einen Schritt weiter gehen Ward et al. [WAB+10] und migrieren ausschließlich die *Workloads* (z.B. eine Java-EAR-Datei), welche auf IBM-Middleware-Komponenten (WebSphere, DB2 etc.) betrieben werden. Dabei werden die relevanten Konfigurations- und Auslastungsinformationen automatisiert extrahiert und darauf basierend ein Image für die Zielumgebung manuell ausgewählt. Auf der virtuellen Maschine werden die gleichen

Middleware-Komponenten installiert, konfiguriert und der Workload bereitgestellt. Die Arbeit zeigt nicht, wie die verschiedenen Komponenten in der Zielumgebung wieder verbunden werden, wenn zum Beispiel eine Java-Anwendung und eine Datenbank migriert werden, und betrachtet nur eine beschränkte Anzahl kommerzieller Middleware- und Hardware-Produkte.

Eine weitere Gruppe von Ansätzen verzichtet auf Images und migriert Komponenten direkt auf Dienste, welche die entsprechende Middleware betreiben. Diese Ansätze sind für Anwendungen geeignet, die keine oder wenige Abhängigkeiten auf die darunterliegende Infrastruktur, das Netzwerk, Betriebssystem und die Middleware-Konfiguration haben. „CloudMIG“ [FH11a] identifiziert mögliche Probleme und nötige Änderungen an der Anwendung, damit sie auf den Plattformdiensten ausgewählter Cloud-Anbieter betrieben werden kann. Dazu wird die Architektur der Anwendung als Modell extrahiert[1] und anhand ihrer Eignung für die Cloud klassifiziert. Nötige Änderungen an der Architektur müssen manuell durchgeführt werden. Frey und Hasselbring [FH11b] erweitern das von ihnen entwickelte CloudMIG um die Modellierung technischer Einschränkungen von Cloud-Anbietern.[2] Anschließend werden der Code und das Architekturmodell auf Verstöße gegen Einschränkungen der potentiellen Zielumgebung geprüft, die ein manuelles Reengineering der Anwendung erforderlich machen. Einen vergleichbaren Ansatz wählt das EU-Projekt „REMICS“ [MBS+10], das aus einer zu migrierenden Anwendung das Architekturmodell extrahiert, welches dann mithilfe von Techniken aus der modellgetriebenen Architektur und Entwicklung (MDA bzw. MDD) für die Cloud adaptiert, erweitert und validiert wird. Die technische Umsetzung der Architektur, also wie die konkreten Komponenten der abstrakten Architektur migriert werden, wird dabei nicht betrachtet. Auf dem Modell von REMICS aufbauend erlaubt das EU-Projekt „MODAClouds“ [ADNM+12] die Anwendung transparent in verschiedenen

[1] Basierend auf den Arbeiten der „Architecture-Driven Modernization Task Force“ bei der Object Management Group [OMG13].

[2] Beispielsweise darf eine Anwendung bei Google App Engine nur bestimmte Teile des Java SDK verwenden [Goo14]

Clouds zu betreiben und zwischen Clouds zu migrieren. Die Migration ist jedoch nur für Anwendungen möglich, die mit den MODAClouds-Werkzeugen generiert wurden und die entsprechenden Abstraktionen nutzen.

Darüber hinaus stellt sich die Frage, wie die Komponenten verteilt werden sollen: Leymann et al. [LFM+11] nutzen annotierte Architekturen und Deployment-Diagramme, um darauf basierend die optimale Verteilung der Anwendung zwischen verschiedenen Clouds zu berechnen und die Anwendung bereitzustellen. Hajjat et al. [HSS+10] bestimmen außerdem die benötigten Firewall-Konfigurationen, damit die Kommunikation der Komponenten wie bisher möglich ist.

Die vorgestellten Arbeiten betrachten Anwendungen auf unterschiedliche Art und Weise, von monolithischen Images bis zu Architekturen. Was bisher nicht existiert, sind Methoden und Werkzeuge, die eine Anwendungsmigration basierend auf Vorgaben bezüglich Anbieter und Verteilung auf technischer Ebene automatisiert durchführen [JAP13, PXW13]. Dies verlangt insbesondere eine feingranulare Enterprise Topologie, welche die Struktur und die Laufzeitinformationen der Anwendungen enthält.

KAPITEL 3

METHODE ZUR MIGRATION VON ANWENDUNGEN

Dieses Kapitel stellt mit „AROMA“ (Automatisierte Migration von Anwendungen) eine Methode zur technischen Durchführung der Migration von Anwendungen vor. Deren Schritte werden durch die Forschungsbeiträge der vorliegenden Arbeit realisiert und automatisiert. Im Gegensatz zu den in Abschnitt 2.7.2 vorgestellten Arbeiten adressiert die AROMA-Methode die technische und automatisierte Durchführung der Migration und nicht nur das Vorgehen für die manuelle Durchführung. Die Automatisierung ist dabei von besonderer Bedeutung, da sie eine kostengünstige Migration sicherstellt [JAP13, PXW13] und dabei weniger zeitaufwändig und fehleranfällig ist als die manuelle Durchführung [OGP03, FLRS12, BBK+14b]. In der in Abschnitt 2.7 vorgestellten Klassifikation von Jamshidi et al. [JAP13] ist die AROMA-Methode somit der Klasse „Durchführung“ und „Evaluation“ zuzuordnen, wie in Abbildung 3.1 dargestellt.

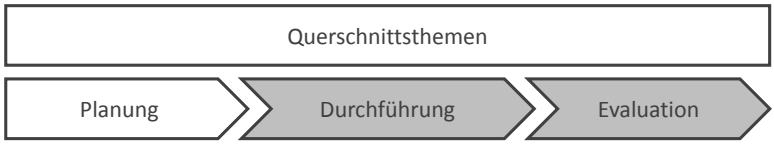

Abbildung 3.1: Klassifikation der AROMA-Methode nach Jamshidi et al. (angelehnt an [JAP13])

Basierend auf den in Abschnitt 3.1 definierten Anforderungen und Herausforderungen behandelt Abschnitt 3.2 die AROMA-Methode und Abschnitt 3.3 deren Varianten. Abschnitt 3.4 diskutiert die AROMA-Methode sowie die Eignung von Anwendungen und Zielumgebungen für eine Anwendungsmigration mit dieser. Die Forschungsbeiträge zur Realisierung der Methode und entsprechende Werkzeugunterstützung zur Automatisierung werden in den nachfolgenden Kapiteln vorgestellt.

3.1 Anforderungen und Herausforderungen

Dieser Abschnitt fasst die verschiedenen Anforderungen und Herausforderungen an die Methode zur Anwendungsmigration zusammen, welche sich aus den vorhergehenden Kapiteln ergeben haben, und erläutert diese.

Anforderung 1 (A-1): Anwendungsfunktionalität erhalten. Die fachliche Funktionalität der Anwendung darf sich durch deren Migration nicht verändern (siehe Definition 2.6 „Anwendungsmigration“). Es soll ausschließlich die Umgebung, in der die Anwendung betrieben wird, gewechselt und alle dafür nötigen Anpassungen durchgeführt werden. Dies ist auch der Schluss von Sneed et al. [SWH10], basierend auf der Erfahrung aus erfolgreichen Migrationsprojekten[1] und dem Prinzip *Separation of Concerns* [Dij76].

Anforderung 2 (A-2): Generizität. Die Migrationsmethode muss auf jede Anwendung angewendet werden können. Die Arten von Komponenten, deren Hersteller, Konfigurationen, Architekturen und Ursprungsumgebun-

[1] „[...] eine Strategie, die in fast allen erfolgreichen Migrationsprojekten zu finden ist: Man ändert so wenig wie möglich, um das primäre Ziel nicht zu gefährden.“ Sneed et al. [SWH10]

gen von bestehenden und zukünftigen Anwendung dürfen deshalb nicht eingeschränkt werden (engl. *open-world assumption*). Insbesondere im Zusammenhang mit der Automatisierung bedeutet dies, dass die Realisierung der Methode erweiterbar sein muss, falls beispielsweise typspezifische Erweiterungen nötig sind.

Anforderung 3 (A-3): Automatisierung. Die AROMA-Methode beschreibt die technische Durchführung einer Anwendungsmigration. Deren Automatisierung durch die weiteren Forschungsbeiträge der vorliegenden Arbeit trägt dazu bei, die Kosten der Anwendungsmigration zu senken und diese somit für mehr Anwendungen rentabel zu machen [JAP13]. Darüber hinaus werden, im Gegensatz zur manuellen Durchführung, Flüchtigkeitsfehler vermieden [Gra86]. In den unterstützenden Werkzeugen kann Expertenwissen gebündelt und somit bei der Durchführung jeder einzelnen Migration wiederverwendet werden.

3.2 AROMA-Methode

Die AROMA-Methode zur technischen Durchführung der Migration von Anwendungen umfasst sieben Schritte. Jeder Schritt wird in diesem Abschnitt detailliert vorgestellt, seine Resultate werden beschrieben und die benötigten Benutzereingaben spezifiziert. Des Weiteren wird auf die anderen Forschungsbeiträge der vorliegenden Arbeit verwiesen, welche die Schritte der Methode realisieren und automatisieren. Die Arbeit zeigt die Umsetzung der Methode für die Zielumgebung Cloud und verwendet ab Schritt 3 zur Beschreibung von Anwendungstopologien den OASIS-Standard TOSCA. Die Methode und ihre Schritte sind jedoch auch auf andere Zielumgebungen anwendbar (vgl. Abschnitt 3.4.2) und mit anderen Sprachen zur Beschreibung von Anwendungstopologien umsetzbar.

Bei der Durchführung der Migration tätigt der Benutzer, typischerweise ein Administrator der Organisation, die entsprechenden Eingaben, woraufhin der jeweilige Schritt automatisiert durchgeführt wird. Die Eingaben sind Entscheidungen des Benutzers, beispielsweise die Auswahl der Anbieter

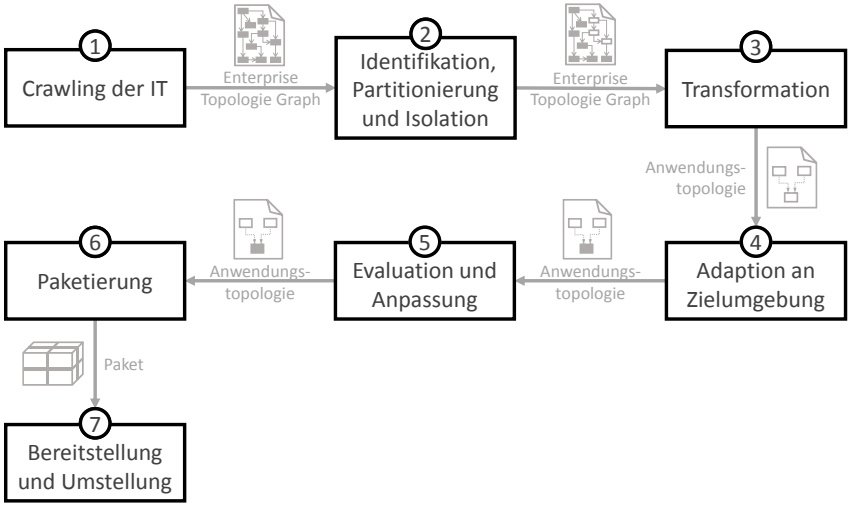

Abbildung 3.2: AROMA-Methode

oder die Verteilung der Anwendungskomponenten. Verwandte Arbeiten, die helfen können, diese Entscheidungen zu treffen, werden im jeweiligen Schritt genannt. Abbildung 3.2 zeigt die sieben Schritte der AROMA-Methode, wobei Rechtecke Daten und die Pfeile den jeweiligen Schritt der Methode repräsentieren. Im Folgenden werden die sieben Schritte der Methode detailliert beschrieben und in Abschnitt 3.3 deren Varianten vorgestellt.

3.2.1 Schritt 1: Crawling der IT

Im ersten Schritt wird die Enterprise Topologie erstellt, die den aktuellen Stand der IT und alle relevanten technischen Details beinhaltet. Alle weiteren Schritte arbeiten ausschließlich mit diesem Modell (bzw. der davon abgeleiteten Anwendungstopologie) und beeinflussen oder verändern die existierenden Anwendungen nicht.

Anforderungen an die Realisierung: Da diese Enterprise Topologie die Grundlage für alle nachfolgenden Schritte ist, muss sie (i) aktuell, (ii) voll-

ständig, (iii) korrekt und (iv) in einer abgestimmten Granularität vorliegen. Die Aktualität ist erforderlich, da sonst möglicherweise eine Anwendung migriert würde, welche in der Realität in der Zwischenzeit verändert wurde. Die Enterprise Topologie muss vollständig und korrekt sein, das heißt, jede relevante Komponente und Relation muss vorhanden sein und die entsprechenden Informationen enthalten. Nicht vorhandene oder nicht korrekt repräsentierte Komponenten und Relationen können zu falschen Ergebnissen und fehlgeschlagenen Migrationen führen. In gleicher Weise muss die Granularität der Enterprise Topologie zwischen den verschiedenen Schritten der AROMA-Methode abgestimmt sein. Dies bedeutet nicht zwangsläufig, dass alle Anwendungsbestandteile in der gleichen Granularität abgebildet werden müssen, sondern dass allen Schritten und Werkzeugen die jeweilige Granularität bekannt ist. Die Granularität wird im Rahmen der Definition des Metamodells für die Repräsentation von Enterprise Topologien diskutiert (vgl. Abschnitt 4.3). Wie in Abschnitt 2.6 und Anforderung A-3 diskutiert wurde, ist für den Anwendungsfall Migration nur eine automatisierte Erstellung der Enterprise Topologie sinnvoll.

Benutzereingabe: Zum Crawling der Enterprise Topologie sind teilweise die Zugangsdaten für die darin enthaltenen Komponenten nötig, um die entsprechenden Informationen extrahieren zu können. Diese müssen von einem Administrator bereitgestellt werden. Außerdem muss eine oder mehrere Einstiegskomponente angegeben werden, von denen ausgehend das Crawling durchgeführt wird.

Forschungsbeitrag: Das daraus resultierende Forschungsproblem und dessen Lösung wird in Kapitel 5 eingehend behandelt und entspricht *Beitrag 3 („Crawling von IT-Instanzmodellen“)*. Außerdem wird ein Metamodell für die Repräsentation eines IT-Instanzmodells und dessen Verarbeitung benötigt. Dazu werden in Kapitel 4 „Enterprise Topologie Graphen“ (ETG) eingeführt, was *Beitrag 2 („Konzept zur Repräsentation und Verarbeitung von IT-Instanzmodellen“)* der vorliegenden Arbeit entspricht.

Ergebnis: Der erste Schritt resultiert in einer vollständigen und korrekten Enterprise Topologie, die den aktuellen Zustand der IT abbildet.

3.2.2 Schritt 2: Identifikation, Partitionierung und Isolation der Anwendung aus der Enterprise Topologie

Durch die Analyse und Bewertung der zuvor erstellen Enterprise Topologie, entscheidet der Benutzer, welche Anwendungen migriert werden sollen. Dazu ordnet er die Komponenten der Enterprise Topologie abstrakten Zielumgebungen zu und kann somit auch eine Anwendung zwischen verschiedenen Zielumgebungen aufteilen (partitionieren). In diesem Schritt sind die Zielumgebungen abstrakt, das heißt, es wird nicht konkretisiert, bei welchem Cloud-Anbieter eine bestimmte Zielumgebung letztendlich betrieben werden soll. Dies geschieht in Schritt 4 bei der Adaption der Anwendung an die konkrete Zielumgebung. Die für eine abstrakte Zielumgebung gewählten Komponenten werden zudem isoliert, das heißt auf Modellebene aus der Ursprungsumgebung herausgelöst.

Aufgrund der großen Anzahl zu migrierender Anwendungen pro Organisation [Sha12] kann die AROMA-Methode mehrere Anwendungen in einem Durchlauf migrieren, indem in diesem Schritt mehrere Anwendungen gewählt, identifiziert und isoliert werden. Alternativ können mehrere Anwendungen auch nacheinander migriert werden, indem die Schritte 2 bis 7, auf der gleichen Enterprise Topologie, mehrmals durchlaufen werden. Dies ist möglich, da in Schritt 1 die Enterprise Topologie der gesamten IT erstellt werden kann, die alle Anwendungen enthält.

Anforderungen an die Realisierung: Den Komponenten mit Anwendungsfunktionalität kommt in diesem Schritt eine besondere Bedeutung zu, da sie den überwiegenden Teil der Logik enthalten, der auch nach der Migration erhalten bleiben muss (Anforderung A-1). Die anderen Komponenten der Enterprise Topologie betreiben die Komponenten mit Anwendungsfunktionalität, spielen aber eine untergeordnete Rolle, da sie stärker standardisiert und daher leichter austauschbar sind. Für den Benutzer soll es möglich sein, die Auswahl der Anwendung basierend auf den Komponenten mit Anwendungsfunktionalität zu treffen, also einer abstrahierten Sicht der gesamten Enterprise Topologie. Ausgehend von der zur Migration gewählten Anwendungsfunktionalität werden die Komponenten identifiziert, welche die zu

migrierende Komponente betreiben. Es wird also der Teil der Enterprise Topologie identifiziert, der für den Betrieb der jeweiligen Komponenten mit Anwendungsfunktionalität nötig ist. Dabei werden diese Komponenten auch isoliert, das heißt von Komponenten getrennt, die nicht für den Betrieb der Komponenten mit Anwendungsfunktionalität nötig sind.

Benutzereingabe: Der Benutzer muss für diesen Schritt die Auswahl der Anwendungen, welche migriert werden sollen, und deren Verteilung spezifizieren. Dies geschieht durch die Zuordnung der Komponenten mit Anwendungsfunktionalität zu abstrakten Zielumgebungen. Ob und welche Anwendungen bzw. deren Komponenten mit Anwendungsfunktionalität migriert werden sollen, wird in verschiedenen verwandten Arbeiten diskutiert [HSS+10, KHSS10, SP11, TUS11, BGRV12, CB12]. Die Verteilung auf Zielumgebungen gruppiert diese Komponenten beispielsweise aufgrund ihrer Netzwerkkommunikation oder Vorgaben der Fachabteilungen. Somit kann der Benutzer definieren, dass eine Anwendung in Zielumgebung A, eine zweite in Zielumgebung B und eine dritte Anwendung nicht migriert werden soll.[1] Andere verwandte Arbeiten [HSS+10, TLMB10, LFM+11, SSSL12, FFH13] adressieren die Durchführung und Optimierung der Verteilungen.

Forschungsbeitrag: Das daraus resultierende Forschungsproblem und dessen Lösung wird in Abschnitt 4.6 betrachtet und entspricht *Beitrag 4 („Identifikation und Isolation der zu migrierenden Anwendung“)* der vorliegenden Arbeit.

Ergebnis: Für jede abstrakte Zielumgebung ergibt dieser Schritt eine Menge von Komponenten, die in diese Zielumgebung migriert werden sollen.

3.2.3 Schritt 3: Transformation in Anwendungstopologie

Für jede abstrakte Zielumgebung, also die Menge von Komponenten, welche in eine bestimmte Zielumgebung migriert werden soll, wird eine Anwendungstopologie erstellt. Die Umsetzung ist von der gewählten Modellierungssprache für Anwendungstopologien abhängig. Verschiedene Ansätze dazu wurden in Abschnitt 2.4 betrachtet.

[1] Letzteres wird spezifiziert, indem die Komponente keiner Zielumgebung zugeordnet wird.

Anforderungen an die Realisierung: Die Transformation erfolgt generisch und unabhängig von der späteren konkreten Zielumgebung. Somit wären die resultierenden Anwendungstopologien grundsätzlich in der Ursprungsumgebung lauffähig, jedoch nicht zwangsläufig in der Zielumgebung. Um später die automatisierte Bereitstellung der Anwendungen zu ermöglichen, muss bei der Transformation sichergestellt werden, dass alle Komponenten, deren Konfiguration, Artefakte und Relationen in der Anwendungstopologie enthalten sind. Außerdem wird die Enterprise Topologie, welche ein Instanzmodell darstellt, bei der Transformation in eine Anwendungstopologie abstrahiert. Das bedeutet, dass Laufzeitinformationen, etwa die IP-Adressen, nicht übertragen werden, weil diese Eigenschaften erst bei der automatisierten Bereitstellung in der Zielumgebung gesetzt werden.

Benutzereingabe: Keine im Kontext der AROMA-Methode. Möglicherweise verlangt die Umsetzung dieses Schrittes in der gewählten Sprache zur Beschreibung von Anwendungstopologien jedoch Benutzereingaben.

Forschungsbeitrag: Im Rahmen der Umsetzung der AROMA-Methode mit TOSCA (Beitrag 5), wird dieser Schritt in Abschnitt 6.3.1 betrachtet.

Ergebnis: In diesem Schritt wurde eine Anwendungstopologie pro abstrakter Zielumgebung erstellt, die aber noch nicht für die Zielumgebung adaptiert wurde.

3.2.4 Schritt 4: Adaption an die konkrete Zielumgebung

Durch die Auswahl eines Anbieters wird die abstrakte Zielumgebung durch den Benutzer konkretisiert. Dieser Schritt passt ausgehend davon die Anwendungstopologie aus dem vorhergehenden Schritt an die konkrete Zielumgebung an.

Die Anpassung an die Zielumgebung wird in der Anwendungstopologie durchgeführt. Dazu können beispielsweise Komponenten ohne Anwendungsfunktionalität (Anforderung A-1) durch kompatible Komponenten aus der Zielumgebung ersetzt und entsprechend konfiguriert werden. Diese Komponenten im unteren Teil der Anwendungstopologie sind überwiegend so standardisiert, dass sie ohne negative Auswirkungen auf die darauf betriebenen Komponenten ersetzt oder adaptiert werden können (vgl. Abschnitt 2.4).

Anforderungen an die Realisierung: Abhängig von den Diensten des gewählten Anbieters kann die Anpassung der Anwendungstopologie an die Zielumgebung auf verschiedenen Ebenen passieren. Bei der Cloud als Zielumgebung kann die Anwendung durch die Nutzung von Plattform- oder Infrastrukturdiensten von den spezifischen Eigenschaften der Cloud als Zielumgebung profitieren. Zum Beispiel kann eine Datenbank als Dienst genutzt werden, was zusätzlich die nichtfunktionalen Eigenschaften der Anwendung verbessert und dem Benutzer Verwaltungsaufgaben wie Backup, Monitoring, Skalierung, Installation von Updates usw. erspart. Alternativ können die entsprechenden Komponenten auch auf einer virtuellen Maschine installiert und konfiguriert werden. Falls die Anwendung in der Ursprungsumgebung auf diese Weise betrieben wurde, reduziert dies das Risiko der Migration, da Plattformdienste häufig Annahmen treffen und Einschränkungen einführen müssen, um beispielsweise Elastizität oder Mehrmandantenfähigkeit zu erreichen. Es ist wichtig anzumerken, dass auch bei der Nutzung eines Infrastrukturdienstes keine VM-Migration stattfindet: Die Anwendung wird basierend auf den in Schritt 1 ermittelten Information neu und automatisiert auf einer virtuellen Maschine aufgesetzt. Somit profitiert die Anwendung aufgrund geringerer Kosten, Verbesserungen beim Management, schnellerer Bereitstellung und Flexibilität von der Cloud. In jedem Fall kann der Benutzer zwischen den verschiedenen Möglichkeiten zur Realisierung wählen.

Benutzereingabe: Der Benutzer wählt als Eingabe dieses Schrittes für jede abstrakte Zielumgebung einen konkreten Anbieter bzw. dessen Dienste aus, zu dem die entsprechenden Komponenten migriert werden sollen. Durch die Auswahl von konkreten Diensten eines Anbieters kann der Benutzer die in den Anforderungen diskutierten Ebenen der Adaption bestimmen, beispielsweise durch Auswahl eines Plattformdienstes anstatt eines Infrastrukturdienstes.

Um einen Anbieter auszuwählen, gibt es Ansätze, welche die technischen Einschränkungen betrachten [FH11b, FFH12], den Aufwand eines Migrationsprojektes abschätzen [TLF+11], für eine gegebene Anwendungstopologie oder Liste benötigter Ressourcen die Kosten optimieren [TLMB10, TUS11, GGRTRC13] und neben den Kosten auch nichtfunktionale Anforderungen in

Betracht ziehen [GVB11, SP11, MR12, ASL13a, ASL13b, FFH13, ASLW14, JVZS+14]. Auch online gibt es eine Reihe von Werkzeugen, die Preisberechnungen und -vergleiche ermöglichen.[1]

Forschungsbeitrag: Das daraus resultierende Forschungsproblem, die Adaption einer Anwendungstopologie an eine konkrete Zielumgebung und die Realisierung der Adaption mit TOSCA als Modellierungssprache für Anwendungstopologien (Beitrag 5), wird in Abschnitt 6.3.2 betrachtet.

Ergebnis: Dieser Schritt resultiert in Anwendungstopologien, die in der entsprechenden konkreten Zielumgebung lauffähig sind.

3.2.5 Schritt 5: Evaluation und manuelle Anpassung

In diesem Schritt hat der Benutzer die Möglichkeit, die Anwendungstopologien zu evaluieren und, falls gewünscht, manuell zu bearbeiten. Somit wird das vorläufige Ergebnis der Migration, die für die Zielumgebung adaptierte Anwendungstopologie, vom Benutzer verifiziert, bevor die Migration in den beiden folgenden Schritten abgeschlossen wird. Im Gegensatz zu den vorhergehenden Schritten, die aufgrund der Benutzereingabe automatisiert durchgeführt werden, ist in diesem Schritt auch die Durchführung manuell.

Anforderungen an die Realisierung: Um dies zu ermöglichen, wird jede Anwendungstopologie in einem für die gewählte Modellierungssprache passenden (grafischen) Modellierungswerkzeug betrachtet. Dadurch können automatisiert getroffene Entscheidungen, wie zum Beispiel einzelne Konfigurationsparameter, angepasst oder Daten ergänzt werden, beispielsweise die Zugangsdaten für den jeweiligen Dienst bzw. Anbieter. Durch die Nutzung einer bestehenden Modellierungssprache für Anwendungstopologien können alle Funktionen der entsprechenden Modellierungswerkzeuge genutzt werden. Denkbar wäre auch die Integration unterstützender Werkzeuge, welche die Anwendung von Mustern erlauben [Wil12, BBKL14c, FLR+14].

Benutzereingabe: Alle vom gewählten (grafischen) Modellierungswerkzeug unterstützten.

[1] Zum Beispiel: Cloudorado (`http://cloudorado.com`), RightScale ShopForCloud (`http://shopforcloud.com`), Aotearoa (`http://aotearoadecisions.appspot.com/`)

Forschungsbeitrag: Im Rahmen der Umsetzung der AROMA-Methode mit TOSCA (Beitrag 5) wird die Integration eines Modellierungswerkzeuges in den Prototypen dieser Arbeit in Abschnitt 6.2 betrachtet.

Ergebnis: Dieser Schritt resultiert in möglicherweise angepassten Anwendungstopologien, die in der entsprechenden konkreten Zielumgebung lauffähig sind.

3.2.6 Schritt 6: Paketierung

Aus der Anwendungstopologie und allen von der Laufzeitumgebung benötigten Artefakten wird ein Paket für die automatisierte Bereitstellung erstellt. Dieses Paket bündelt die serialisierte Anwendungstopologie (z.B. in XML oder JSON) und die anderen Artefakte (z.B. Konfigurationsdateien oder Softwarepakete). Dabei sind das Paketformat und die dafür benötigten Metainformationen abhängig von der verwendeten Modellierungssprache für Anwendungstopologien. Zum Beispiel paktiert TOSCA als CSAR (vgl. Abschnitt 2.4.3.3).

Benutzereingabe: In diesem Schritt kann der Benutzer zwischen verschiedenen Arten wählen, wie ihm die Pakete bereitgestellt werden: (i) in einem Paket pro konkreter Zielumgebung oder (ii) einem Paket für alle zur Migration gewählten Komponenten. Dies ist abhängig davon, ob der Benutzer die Anwendung pro Zielumgebung von einer entsprechenden Laufzeitumgebung oder von einer zentralen Laufzeitumgebung betreiben lassen will. Ersteres erlaubt es beispielsweise gleichzeitig mit der Migration den Betrieb von Teilen der Anwendung auszulagern. Die zweite Option stellt durch eine zentrale Laufzeitumgebung, die alle Komponenten in allen Zielumgebungen verwaltet, eine globale Sicht auf die verteilte Anwendung sicher. Andererseits muss die verwendete Modellierungssprache für Anwendungstopologien und Laufzeitumgebung die gewählte Art der Paketierung auch unterstützen.

Anforderungen an die Realisierung: Die Realisierung ist abhängig von der gewählten Art der Paketierung: (i) Bei der Bereitstellung eines Paketes pro konkreter Zielumgebung muss sichergestellt werden, dass Abhängigkeiten zwischen unterschiedlichen Paketen beachtet werden, die nicht in der

Anwendungstopologie der jeweiligen Zielumgebung enthalten sind. Zum Beispiel müssen die Adresse und die Zugangsdaten einer Datenbankkomponente in einem Paket den darauf zugreifenden Diensten in einem anderen Paket zur Verfügung gestellt werden. Falls die für die Anwendungstopologie gewählte Modellierungssprache paketübergreifende Abhängigkeiten nicht unterstützt, werden die Abhängigkeiten dem Benutzer angezeigt, so dass der diese manuell auflösen kann. In Zukunft wäre es auch denkbar, diese Option dahingehend zu erweitern, dass die Anwendungstopologie für jede Zielumgebung eine andere Sprache nutzt.

(ii) Die Bereitstellung eines Paketes für alle zu migrierenden Komponenten setzt voraus, dass die gewählte Modellierungssprache für Anwendungstopologien und die entsprechende Laufzeitumgebung in einer Anwendungstopologie Komponenten verschiedener Anbieter unterstützen. Dies hat den Vorteil, dass Abhängigkeiten zwischen Komponenten in verschiedenen Zielumgebungen zentral verwaltet und aufgelöst werden können.

Ergebnis: Das Ergebnis dieses Schrittes ist ein Paket pro konkrete Zielumgebung, falls der Benutzer Option (i) gewählt hat, oder ein Paket für alle zu migrierenden Komponenten, falls der Benutzer Option (ii) gewählt hat.

3.2.7 Schritt 7: Bereitstellung und Umstellung

In diesem Schritt wird die Migration mit der Durchführung von (i) Bereitstellung, (ii) Test, (iii) Datenmigration, (iv) Umstellung und (v) Außerbetriebnahme in der Ursprungsumgebung abgeschlossen. Im Folgenden wird auf diese fünf Teilschritte eingegangen und die Anforderungen an ihre Realisierung diskutiert.

(i) *Bereitstellung.* Mithilfe der entsprechenden Laufzeitumgebung werden die im vorhergehenden Schritt erstellten Pakete automatisiert bereitgestellt. Um dies zu ermöglichen, muss bei der Wahl der Modellierungssprache der Anwendungstopologie auf die Verfügbarkeit einer Laufzeitumgebung geachtet werden, die eine automatisierte Bereitstellung erlaubt.

(ii) *Test.* Die Instanz der Anwendung in der Zielumgebung wird nach deren Bereitstellung getestet. Alternativ kann auch eine separate, aber mit der

eigentlichen Anwendungstopologie übereinstimmende Instanz als Testumgebung genutzt werden, deren automatisierte Bereitstellung mit geringem zusätzlichem Aufwand realisierbar ist. Insbesondere Regressionstests erlauben es, die funktionale Äquivalenz der Anwendung zu prüfen [IEE90]. Auch das Verhalten der Anwendung unter Last sollte getestet und gegebenenfalls die Konfiguration der Komponenten entsprechend angepasst werden.

(iii) *Datenmigration.* Die Anwendung in der Zielumgebung enthält zu diesem Zeitpunkt nur die Konfiguration der Komponenten, nicht jedoch deren Anwendungsdaten. Daten aus Datenbanken können mit den entsprechenden Werkzeugen der Hersteller direkt migriert werden, auch falls in der Zielumgebung eine andere Lösung für die Datenhaltung verwendet wird. Darüber hinaus gibt es verwandte Arbeiten, welche die Auswahl des Dienstes für die Datenhaltung unterstützen [SKLU11, SAT+13], und Muster, welche die Migration von Daten unter Berücksichtigung von beispielsweise Vertraulichkeit [SAB+13] beschreiben. Andere Daten der Anwendung, die zum Beispiel in Dateien verwaltet werden, müssen meist manuell oder mit anwendungsspezifischen Werkzeugen migriert werden.

(iv) *Umstellung.* Nach der Datenmigration ist die Anwendung bereit für die produktive Nutzung. Um von der Anwendungsinstanz in der Ursprungsumgebung auf die in der Zielumgebung umzustellen, werden die entsprechenden Einstiegspunkte in die Anwendung aktualisiert – zum Beispiel indem der DNS-Eintrag geändert wird, der bisher auf die Anwendung in der Ursprungsumgebung zeigt, oder die Endpunktinformationen im Geschäftsprozess oder ESB aktualisiert werden. Falls eine Migration ohne Unterbrechung der Verfügbarkeit durchgeführt werden soll, müssen die Datenmigration und Umstellung koordiniert und gleichzeitig geschehen. Das Muster *Forklift Migration* von Fehling et al. [FLR+13] beschreibt beispielsweise den Ablauf, wie eine solche Migration durchgeführt werden könnte.

(v) *Außerbetriebnahme in Ursprungsumgebung.* Sobald die Instanz der migrierten Anwendung in der Zielumgebung genutzt wird, können die in der Ursprungsumgebung nicht mehr benötigten Komponenten außer Betrieb genommen werden. Ob eine Komponente noch benötigt wird oder nicht, kann auf Basis der Enterprise Topologie analysiert werden.

Forschungsbeitrag: Die Migrationsmethode ist unabhängig von der gewählten Modellierungssprache der Anwendungstopologie. Für die Realisierung der AROMA-Methode wird im Rahmen der vorliegenden Arbeit TOSCA verwendet. Der erste Teilschritt, die automatisierte Bereitstellung mit TOSCA, wird in Kapitel 6 betrachtet. Die Teilschritte Test, Datenmigration, Umstellung und Außerbetriebnahme werden nicht im Rahmen der vorliegenden Arbeit behandelt. Für jeden dieser Teilschritte existieren jedoch verwandte Arbeiten, welche diese Teilschritte adressieren.

Ergebnis: Das Ergebnis dieses Schrittes und auch der gesamten AROMA-Methode ist, dass die migrierten Anwendungen in den Zielumgebungen bereitgestellt und getestet wurden. Darüber hinaus wurden die Daten übertragen, die Nutzung der Anwendung auf die neue Instanz umgestellt und die nicht mehr benötigten Komponenten in der Ursprungsumgebung außer Betrieb genommen.

3.3 Varianten der Methode

Abhängig von den Details der geplanten Migration, wie zum Beispiel um welche Art von Anwendung es sich handelt oder ob nur eine Anwendung migriert werden soll, kann die Durchführung einer Variante der Methode angebracht sein. Dieser Abschnitt stellt zwei Varianten der zuvor eingeführten AROMA-Methode vor und diskutiert, wann ihre Verwendung sinnvoll ist.

3.3.1 Variante 1: Selektive Erstellung der Enterprise Topologie

Anstatt zuerst die gesamte Enterprise Topologie zu erstellen, kann Schritt 1 der AROMA-Methode auch selektiv durchgeführt werden, falls schon entschieden wurde, welche Anwendung migriert werden soll. Dazu werden, wie in Abbildung 3.3 hervorgehoben, die Schritte 1 und 2 der AROMA-Methode angepasst. In Schritt 1 wird, ausgehend von der ausgewählten Anwendung, die Erstellung der Enterprise Topologie so durchgeführt, dass nur die für die gewählte Anwendung relevanten Komponenten und Relationen enthalten sind. Diese Einschränkung reduziert den Aufwand und die Laufzeit der

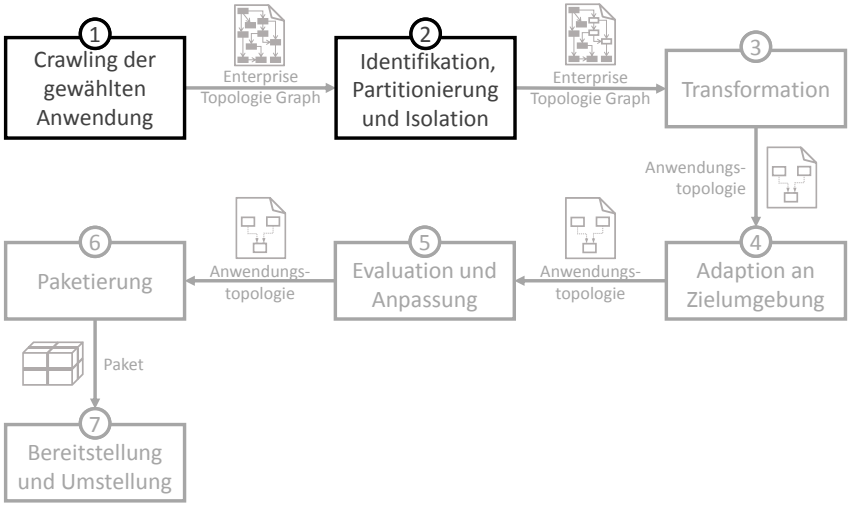

Abbildung 3.3: AROMA-Methode Variante 1

Erstellung der Enterprise Topologie und erlaubt somit auch eine schnellere Migration einzelner Anwendungen. Die Herausforderung dabei ist es, trotzdem alle Abhängigkeiten anderer Komponenten auf die Komponenten der zu migrierenden Anwendung zu identifizieren, so dass die nachfolgenden Schritte korrekt durchgeführt werden können. Das daraus resultierende Forschungsproblem, die *Selektive Erstellung der Enterprise Topologie einer Anwendung*, und dessen Lösung werden in Abschnitt 5.1.5.4 betrachtet und sind ein Teil von *Beitrag 3 („Crawling von IT-Instanzmodellen“)*. In Schritt 2 muss in dieser Variante keine Anwendung gewählt werden, da keine Identifikation nötig ist. Die Partitionierung und Isolation der Anwendung muss jedoch wie beschrieben durchgeführt werden.

3.3.2 Variante 2: Migration geschäftsprozessbasierter Anwendungen

Der Einsatz von Geschäftsprozessen zur Orchestrierung von Diensten hat das Ziel, flexibel auf Veränderungen des Marktes reagieren zu können. Diese Art

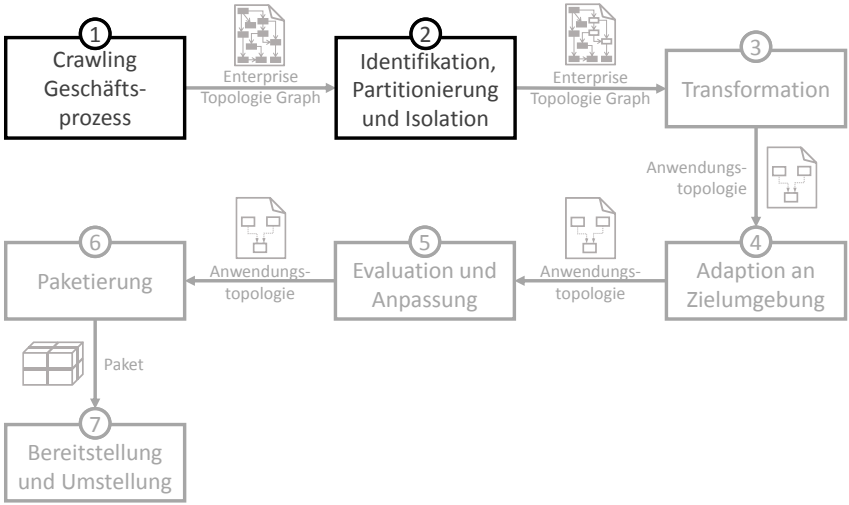

Abbildung 3.4: AROMA-Methode Variante 2

der Integration verschiedener Dienste, Anwendungen und Organisationen stellt oft auch die Schnittstelle zwischen Fachabteilungen und IT dar. In Enterprise Topologien sind Geschäftsprozesse eine Komponente wie jede andere. Aufgrund ihrer Bedeutung wird aber diese Variante der Methode definiert, welche die Migration von Anwendungen unterstützt, die durch einen Geschäftsprozess und eine Menge von Diensten implementiert sind. Dazu verwendet diese Variante der AROMA-Methode eine auf Geschäftsprozesse abgestimmte Visualisierung, führt nur eine selektive Erstellung der Enterprise Topologie durch (vgl. Variante 1) und behandelt den zu migrierenden Geschäftsprozess gesondert. Wie in Abbildung 3.4 hervorgehoben, wird zuerst der zu migrierende Geschäftsprozess ausgewählt und davon ausgehend eine selektive Erstellung der Enterprise Topologie durchgeführt. Der zur Migration gewählte Geschäftsprozess wird für den Benutzer nun so visualisiert, dass dieser die Verteilung auf Ebene der Aktivitäten des Geschäftsprozesses definieren kann, das heißt feingranularer als bei der komponentenbasierten Auswahl der normalen Methode. Darüber hinaus kann eine Sicht auf den

Geschäftsprozess angeboten werden, welche nur Aktivitäten zeigt, die Relationen zu andere Komponenten haben, zum Beispiel da sie Dienste aufrufen. Die gewählte Verteilung der Aktivitäten und die daraus abgeleitete Verteilung der vom Geschäftsprozess genutzten Komponenten ist die Eingabe für Schritt 2, der (wie alle nachfolgenden Schritte) unverändert durchgeführt wird. Das daraus resultierende Forschungsproblem, die automatisierte Migration von geschäftsprozessbasierten Anwendungen, und dessen Lösung werden in Abschnitt 7.4.3 betrachtet und sind ein Teil von *Beitrag 1*.

Für die Behandlung der Geschäftsprozesskomponente selbst gibt es wiederum zwei Untervarianten:

3.3.2.1 Variante 2a: Geschäftsprozess wird nicht migriert

In dieser Untervariante bleibt der Geschäftsprozess in der Ursprungsumgebung, es werden also nur die durch den Prozess orchestrierten Dienste migriert. Nach der Bereitstellung werden im Geschäftsprozess die entsprechenden Endpunktinformationen angepasst. Somit wird der Geschäftsprozess in der Ursprungsumgebung zur Integration der migrierten Dienste in verschiedenen Zielumgebungen genutzt und das Prozesswissen der Organisation kann intern bleiben.

3.3.2.2 Variante 2b: Verteilung und Migration des Geschäftsprozesses

Alternativ kann auch der Geschäftsprozess, genauso wie die migrierte Anwendung, aufgeteilt und migriert werden. Für einen Geschäftsprozess in BPEL zeigen Khalaf et al. [KL06, KKL08], wie man diesen, basierend auf der vom Benutzer bestimmten Verteilung, in eine Menge von Geschäftsprozessen aufteilt. Diese Geschäftsprozesse können in den unterschiedlichen Zielumgebungen betrieben werden und realisieren zusammen die gleiche Anwendungsfunktionalität wie der ursprüngliche Geschäftsprozess. Falls mehrere Teile des ursprünglichen Geschäftsprozesses in die gleiche Zielumgebung migriert werden sollen, können diese zur Optimierung wieder in einen Geschäftsprozesse zusammengeführt werden [WKL11].

3.4 Diskussion und Zusammenfassung

Die vorgestellte Methode ermöglicht die Migration existierender Anwendungen von der Ursprungsumgebung in die Zielumgebung, ohne die grundsätzliche Art und Weise, wie diese gebaut wurden, zu verändern. Dies erlaubt es der Anwendung von der Zielumgebung zu profitieren, beispielsweise durch eine Reduzierung der Betriebskosten oder durch die Verbesserung der nichtfunktionalen Anforderungen, ohne die Funktionalität der Anwendung zu verändern (Anforderung A-1). Die AROMA-Methode umfasst die technische Durchführung einer Anwendungsmigration und sieht deren Automatisierung vor (Anforderung A-3). Die anderen Forschungsbeiträge der vorliegenden Arbeit, welche in den nachfolgenden Kapiteln vorgestellt werden, realisieren und automatisieren die Schritte der AROMA-Methode.

Ein entscheidender Faktor für die erfolgreiche Durchführung ist die Qualität der aus Schritt 1 resultierenden Enterprise Topologie, die Informationen über alle IT-Komponenten einer Organisation enthält. Bis Schritt 3 nutzt die AROMA-Methode die Enterprise Topologie als zentrales Modell, anschließend eine daraus abgeleitete Anwendungstopologie. Die gesamte Anwendungsmigration stützt sich damit auf dieses IT-Instanzmodell, weshalb die Sicherstellung der in Abschnitt 3.2.1 diskutierten Anforderungen fundamental wichtig für die Migration ist. Aus diesem Grund wird die Erstellung der Enterprise Topologie in Kapitel 5 ausführlich betrachtet und stellt einen Forschungsschwerpunkt der vorliegenden Arbeit dar.

Die Vorteile der Durchführung der Migration an einem Modell sollen hier besonders herausgehoben werden: Die AROMA-Methode verändert bis Schritt 7 die Anwendung in der Ursprungsumgebung nicht. Dies ermöglicht es dem Benutzer, verschiedene Anbieter und Dienste zu evaluieren und die am besten geeignete Lösung zu wählen. Damit folgt die vorliegende Arbeit beispielsweise dem Referenzprozess von Ackermann et al. [AGW05], der zuerst das Modell der Anwendung komplett für die Zielumgebung adaptiert und anschließend bereitstellt. Erst danach werden die Daten migriert und schlussendlich die Zugriffspfade auf die neue Version umgestellt.

Durch die AROMA-Methode können somit die nicht-technischen und nicht-automatisierten Vorgehensmodelle, welche in Abschnitt 2.7.2 vorgestellt wurden, umgesetzt werden, anstatt wie bisher die Migration manuell durchzuführen. Die Vorgehensmodelle aus Abschnitt 2.7.2 bilden dabei den organisatorischen Rahmen, der zum Beispiel die Benutzereingaben bereitstellt. Die Umsetzung der in Abschnitt 2.7.1 eingeführten Migrationsstrategie 1 („Migration der kompletten Anwendung“) und Migrationsstrategie 2 („Migration von Teilen der Anwendung“) ist mit der AROMA-Methode möglich, indem die Eingabe von Schritt 2 entsprechend gewählt wird. Die Migration ohne Unterbrechung der Verfügbarkeit (Migrationsstrategie 3) muss in Schritt 7 gewährleistet werden, dessen Realisierung auf verwandten Arbeiten basiert und nicht Beitrag der vorliegenden Arbeit ist. Prinzipiell muss dabei die Umstellung auf die Zielumgebung und Datenmigration so durchgeführt werden, dass die Anwendung für den Benutzer stets verfügbar ist [FLR+13].

Im Folgenden diskutiert Abschnitt 3.4.1 die Eignung von Anwendungen für die Migration mit der AROMA-Methode und Abschnitt 3.4.2 betrachtet die Anforderungen an potentielle Zielumgebungen der Migration.

3.4.1 Für die Migration geeignete Anwendungen

Die AROMA-Methode und die genutzten Datenmodelle sind so generisch ausgelegt, dass generell jede Art von Anwendung damit migriert werden kann (Anforderung A-2), jedoch mit unterschiedlichem Aufwand. Die Einhaltung von Standards und Best Practices sowie eine *saubere Programmierung und Design* der Anwendung im Sinne des Software Engineerings machen die Migration, genauso wie andere Verwaltungsarbeiten auch, einfacher [DMT13]. Anforderungen dafür beschreibt unter anderen Wiggins [Wig12], beispielsweise Zugangsdaten und andere Einstellungen in separate Datenbanken oder Dateien auszulagern, anstatt diese im Code einzubetten.

Zusammengesetzte Anwendungen [ML10] mit expliziten Abhängigkeiten zu anderen Komponenten sind generell besser geeignet als monolithische Blöcke. Grund dafür ist, dass zusammengesetzte Anwendungen eine klar definierte Trennung zwischen verschiedenen Komponenten aufweisen. Zu

dieser Art von Anwendungen zählen auch SOA-Anwendungen, die für eine Migration, beispielsweise in die Cloud, besonders geeignet sind [FLR+13]. Geschäftsprozesse, welche Dienste über eine klar definierte Schnittstelle zu einer neuen Geschäftsfunktionalität komponieren (siehe Abschnitt 2.1), erfüllen diese Anforderung auch. Wie in Variante 2 („Migration geschäftsprozessbasierter Anwendungen“) beschrieben, ist es so beispielsweise möglich die durch den Geschäftsprozess aufgerufenen Dienste in eine andere Umgebung zu migrieren und dabei nur die Endpunktinformationen im Geschäftsprozess zu ändern.

3.4.2 Für die Migration geeignete Zielumgebungen

Eine Zielumgebung für eine mit der AROMA-Methode migrierten Anwendung kann jede IT-Umgebung sein, in welcher die Anwendungstopologie bereitgestellt werden kann (Schritt 7). Da ein wichtiger Aspekt der AROMA-Methode die Automatisierung der Migration ist (Anforderung A-3), betrachtet dieser Abschnitt die Voraussetzungen für die automatisierte Bereitstellung der Anwendung in der Zielumgebung. Eine manuelle Bereitstellung durch einen Administrator ist alternativ möglich, soll aber vermieden werden. Die automatisierte Bereitstellung ist abhängig von der in der Umsetzung verwendeten Sprache zur Beschreibung von Anwendungstopologien. In der vorliegenden Arbeit wurde dafür TOSCA gewählt (vgl. Kapitel 6). Die Voraussetzung für die automatisierte Bereitstellung einer Anwendung mit TOSCA ist, dass die Zielumgebung es ermöglicht, die Anwendungsbestandteile Infrastruktur und Middleware (vgl. Abschnitt 2.2) programmatisch zu verwalten, das heißt, dass sie diese über eine API oder einen Dienst anbietet. Die auf diesen Bestandteilen aufbauenden Komponenten werden dann von der entsprechenden Laufzeitumgebung, wie in der Anwendungstopologie beschrieben, darauf betrieben.

Im Folgenden werden „Cloud“ und „Grid“ [BYV+09] als Beispiele für in Frage kommende Zielumgebungen betrachtet: Die Cloud-Eigenschaften Selbstbedienung und Automatisierung (vgl. Abschnitt 2.3.1) verlangen von einer Cloud implizit, eine API oder einen Dienst zu deren Verwaltung anzu-

bieten. Somit ist die Cloud eine mögliche Zielumgebung für die Migration von Anwendungen mit der AROMA-Methode. Virtualisierte Umgebungen, auch diejenigen, die nach der Definition keine Cloud sind, bieten normalerweise eine API oder einen Dienst für die programmatische Verwaltung ihrer Ressourcen. Auch die unter dem Begriff „Grid Computing“ zusammengefassten Zielumgebungen erfüllen diese Anforderung, da die Steuerung des Grids dem Benutzer über APIs oder Dienste ermöglicht wird [FZRL08, Ley09].

KAPITEL 4

ENTERPRISE TOPOLOGIE GRAPH

Die Untersuchung der verwandten Arbeiten in Abschnitt 2.5 hat gezeigt, dass bisher kein Metamodell für eine (i) technisch detaillierte, (ii) alle Arten von Anwendungsbestandteilen umfassende, (iii) erweiterbare, (iv) für verschiedene Anwendungsfälle zugängliche, (v) anwendungsübergreifende und (vi) formale Enterprise Topologie existiert. Mit „Enterprise Topologie Graphen“ (ETG) führt dieses Kapitel ein Konzept für die formale Repräsentation von Enterprise Topologien und deren Verarbeitung ein. Ein ETG ist ein Graph, der einen feingranularen Schnappschuss der gesamten IT, das heißt aller Anwendungen, abbildet. Dies beinhaltet alle Komponenten, zum Beispiel Prozesse, Dienste, Software, Middleware, Infrastruktur und beliebige Arten von Cloud-Diensten, die als Knoten dargestellt werden. Die Kanten repräsentieren die Relationen zwischen den Komponenten, zum Beispiel, dass eine Komponente von einer anderen betrieben wird.

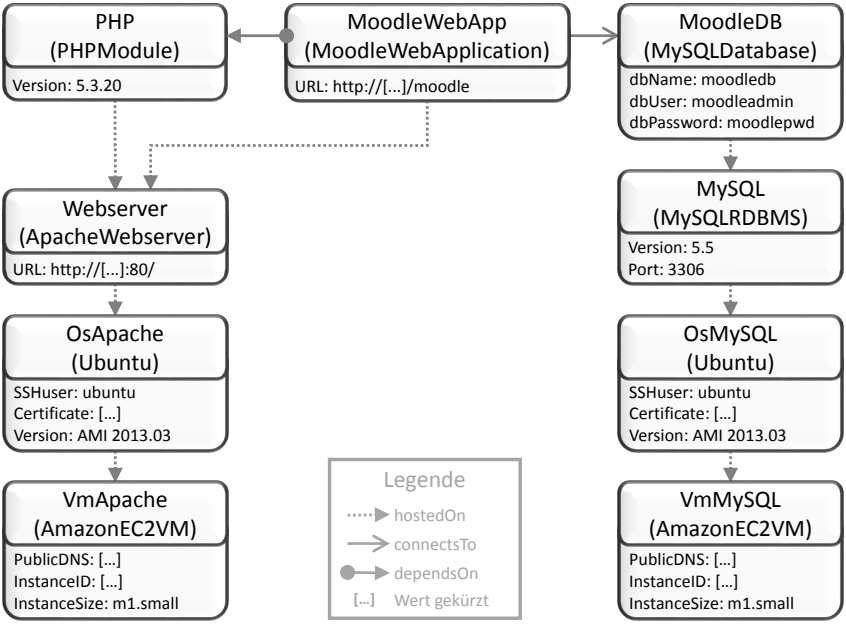

Abbildung 4.1: Ausschnitt eines Beispiel-ETGs der Anwendung „Moodle“

Abbildung 4.1 zeigt einen ETG der Schulverwaltungs- und Lernanwendung „Moodle“ [Moo14], die auf einem LAMP-Stack (Linux, Apache, MySQL, PHP) betrieben wird. Zur besseren Illustration wird hier nur eine Anwendung und somit ein kleiner Ausschnitt des gesamten ETGs gezeigt, der je nach Größe der IT viele Tausende Anwendungen und somit noch mehr Komponenten enthalten kann. Zur Darstellung von ETGs, wie zum Beispiel in Abbildung 4.1, wird die visuelle Notation Vino4TOSCA [BBK+12] verwendet. Diese stellt Knoten als abgerundete Rechtecke dar, die im oberen Bereich den Namen des Knotens und darunter dessen Typ in Klammern enthalten. Die Eigenschaften des Knotens werden im unteren Bereich des jeweiligen Rechtecks dargestellt. Die Abbildung zeigt auch, wie eine Anwendung aufgeteilt in ihre einzelnen technischen Komponenten repräsentiert werden kann, anstatt als monolithischer Block.

Dieses Kapitel führt das Metamodell für Enterprise Topologie Graphen in Abschnitt 4.1 ein und definiert diese daraufhin in Abschnitt 4.2 formal. Die Granularität von ETGs wird in Abschnitt 4.3 diskutiert. Auf dem Metamodell aufbauend führen die Abschnitte 4.4, 4.5 und 4.6 Operationen zur Verarbeitung von ETGs ein, welche im Verlauf dieser Arbeit verwendet werden. Neben den hier vorgestellten wurden in [BFL+12] und [BLNS12] weitere ETG-Operationen definiert. Abschnitt 4.6 stellt ein Konzept für die Identifikation und Isolation der zu migrierenden Komponenten aus dem ETG vor, welches Schritt 2 der AROMA-Methode realisiert. Abschnitt 4.7 fasst das Kapitel zusammen und diskutiert das eingeführte Metamodell zur Repräsentation und Verarbeitung von IT-Instanzmodellen.

4.1 ETG-Metamodell

Dieser Abschnitt gibt einen Überblick über das in Abbildung 4.2 dargestellte ETG-Metamodell. Ein *ETG* enthält eine Menge von *Elementen*, was den *Komponenten* und *Relationen* eines ETGs entspricht. Dabei kann ein Element in keinem, einen oder mehreren Segmenten enthalten sein. Ein *Segment* gruppiert somit eine Teilmenge der im ETG enthaltenen Elemente anhand verschiedener Aspekte. Jedem Element kann eine beliebige Anzahl

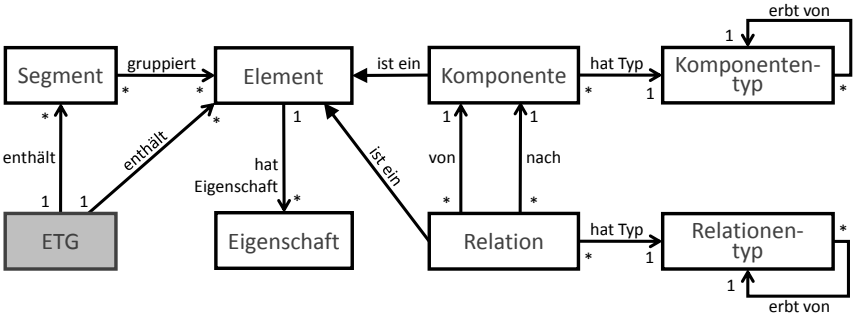

Abbildung 4.2: Enterprise Topologie Graph Metamodell

von *Eigenschaften*, in Form von Name-Wert-Paaren, zugeordnet werden. Damit können Laufzeitinformationen (z.B. IP-Adresse), Konfigurationsdetails (z.B. Version oder Hardwarespezifikation) oder die Implementierung der Komponenten im ETG repräsentiert werden. Die eigentliche Semantik der Komponenten und Relationen wird durch ihre Typen definiert, die in Abbildung 4.2 auf der rechten Seite dargestellt werden. *Komponententypen* repräsentieren die Semantik von einer Klasse von Komponenten, beispielsweise eine PHP-Anwendung, eine bestimmte Art von Webserver, Datenbank, virtueller Maschine oder Netzwerkkomponente. *Relationentypen* stehen für die Semantik der logischen, funktionalen und physikalischen Relationen zwischen zwei Komponenten. Damit kann ausgedrückt werden, dass eine Komponente auf einer anderen Komponente installiert wird, eine Abhängigkeit hat oder mit dieser kommuniziert. Ein Typ kann von einem anderen Typ *erben*. Beispielsweise erbt *Apache Webserver* vom Komponententyp *Webserver* und eine *JDBC-Connection* vom Relationentyp *connectsTo*. Ein vergleichbares Vererbungsprinzip ist beispielsweise im TOSCA-Standard [OAS13b] vorgesehen (vgl. Abschnitt 2.4.3).

4.2 Definition von Enterprise Topologie Graphen

Dieser Abschnitt definiert, dem ETG-Metamodell in Abschnitt 4.1 folgend, den ETG formal.

Definition 4.1 (Enterprise Topologie Graph)**.** Ein Enterprise Topologie Graph ist ein gerichteter, möglicherweise zyklischer Graph [Die05], dessen Knoten gefärbt und dessen Kanten gewichtet sind. Sei $\mathcal{ETG}$ die Menge aller ETGs, sei $t \in \mathbb{N}_0$ der Indikator für dessen Version oder den Zeitpunkt wann die IT im abgebildeten Zustand war, dann wird ein ETG $etg_t \in \mathcal{ETG}$ definiert als 12-Tupel:

$$etg_t = (K, R, von, nach, hatEigenschaften, S, KT, RT, typK, typR, erbtVonKT, erbtVonRT)$$

Dabei ist:

- K die Menge aller Komponenten (siehe Definition 4.2),
- R die Menge aller Relationen (siehe Definition 4.3),
- *von* die Abbildung, die einer Relation die Komponente, von der diese ausgeht, zuordnet (siehe Definition 4.4),
- *nach* die Abbildung, die einer Relation die Komponente, an der diese eingeht, zuordnet (siehe Definition 4.4),
- *hatEigenschaften* die Abbildung, welche jeder Komponente und Relation ihre Eigenschaften zuordnet (siehe Definition 4.15),
- S die Menge aller Segmente (siehe Definition 4.18),
- KT die Menge aller Komponententypen (siehe Definition 4.6),
- RT die Menge aller Relationentypen (siehe Definition 4.7),
- *typK* die Abbildung, welche jeder Komponente einen Typ zuordnet (siehe Definition 4.10),
- *typR* die Abbildung, die jeder Relation einen Typ zuordnet (siehe Definition 4.11),
- *erbtVonKT* die Abbildung, die einem Komponententyp den Typ zuordnet, von dem dieser abgeleitet wurde (siehe Definition 4.8) und
- *erbtVonRT* die Abbildung, die einem Relationentyp den Typ zuordnet, von dem dieser abgeleitet wurde (siehe Definition 4.9).

In den nachfolgenden Abschnitten werden die einzelnen Bestandteile des ETGs definiert und deren Eigenschaften diskutiert. Wo im Folgenden der Zeitpunkt t nicht relevant ist, wird vereinfacht *etg* geschrieben.

4.2.1 Komponenten

Die Knoten des ETGs repräsentieren beliebige physikalische oder logische Komponenten der IT. Beispiele für Komponenten in Abbildung 4.1 sind die virtuelle Maschine, der Webserver oder das PHP-Modul. Komponenten im Kontext des ETGs sind wie folgt definiert:

Definition 4.2 (Komponenten)**.** Die endliche Menge $K = \{k_1, k_2, ..., k_n\}$ enthält alle Komponenten k_i eines ETGs.

4.2.2 Relationen

Im ETG können mehrere Relationen in gleicher Richtung zwischen zwei Komponenten existieren [Die05]. Diese Relationen können sich, müssen sich aber nicht in ihren Typen oder Eigenschaften unterscheiden. Durch die Verwendung von gerichteten Kanten zeigen diese, in welche Richtung die Relation zu lesen ist und wie die Semantik der Relation, zum Beispiel *ist installiert auf*, zu verstehen ist. Der ETG kann Zyklen enthalten, auch Zyklen mit Relationen des gleichen Typs.

Definition 4.3 (Relationen)**.** Die endliche Menge $R = \{r_1, r_2, ..., r_n\}$ enthält alle Relationen r_i eines ETGs. Diese sind gerichtet.

Definition 4.4 (Relationen von/nach)**.** Sei Relation r_i eine gerichtete Relation des ETGs, dann wird ihre Quell- und Zielkomponente durch die Funktionen $von(r_i)$ und $nach(r_i)$ bestimmt. Die Funktionen *von* und *nach* sind dabei wie folgt definiert:

$$von : R \rightarrow K$$
$$nach : R \rightarrow K$$

Zur Bestimmung der ausgehenden und eingehenden Relationen einer Komponente werden, basierend auf Definition 4.4, die beiden Hilfsfunktionen *eingehendeRelationen* und *ausgehendeRelationen* definiert.

Definition 4.5 (Ein-/ausgehende Relationen)**.** Sei K die Menge aller Komponenten eines ETGs, sei R die Menge aller Relationen dieses ETGs, dann sind die Funktionen *eingehendeRelationen* und *ausgehendeRelationen* definiert als:

$$ausgehendeRelationen: K \to \wp(R),\ k \mapsto \{r \mid von(r) = k\}$$
$$eingehendeRelationen: K \to \wp(R),\ k \mapsto \{r \mid nach(r) = k\}$$

4.2.3 Typen

Die Komponententypen und Relationentypen repräsentieren die Semantik der Elemente des ETGs. Graphentheoretisch kann die Typisierung von Komponenten und Relationen auf die Färbung und Gewichtung der Knoten und Kanten eines Graphen abgebildet werden.

Definition 4.6 (Komponententypen)**.** Die endliche Menge aller Typen von Komponenten kt_i eines ETGs ist definiert als $KT = \{kt_1, kt_2, ..., kt_n\}$.

Definition 4.7 (Relationentypen)**.** Die endliche Menge aller Typen von Relationen rt_i eines ETGs ist definiert als $RT = \{rt_1, rt_2, ..., rt_n\}$.

Um eine möglichst genaue Typisierung der Komponenten und Relationen zu ermöglichen, können Typen von anderen Typen erben und somit deren Semantik verfeinern. Mehrfachvererbung ist nicht vorgesehen.

Definition 4.8 (Vererbung von Komponententypen)**.** Sei KT die endliche Menge aller Komponententypen eines ETGs, dann gibt die Funktion $erbtVonKT(kt_a) = kt_b$ an, dass Komponententyp kt_a von Komponententyp kt_b erbt. Die Funktion *erbtVonKT* ist dabei definiert als:

$$erbtVonKT: KT \to KT \cup \{\bot\}$$

Definition 4.9 (Vererbung von Relationentypen)**.** Sei RT die endliche Menge aller Typen von Relationen eines ETGs, dann gibt die Funktion $erbtVonRT(rt_a) = rt_b$ an, dass Relationentyp rt_a von Relationentyp rt_b erbt. Die Funktion *erbtVonRT* ist dabei definiert als:

$$erbtVonRT : RT \rightarrow RT \cup \{\perp\}$$

Die Zuweisung der Typen von Komponenten und Relationen wird durch die beiden Funktionen *typK* und *typR* repräsentiert.

Definition 4.10 (Zuweisung von Komponententypen)**.** Sei K die Menge aller Komponenten eines ETGs, sei KT die Menge aller Komponententypen dieses ETGs, dann ist die Funktion *typK* definiert als:

$$typK : K \rightarrow KT$$

Definition 4.11 (Zuweisung von Relationentypen)**.** Sei R die Menge aller Relationen eines ETGs, sei RT die Menge aller Relationentypen dieses ETGs, dann ist die Funktion *typR* definiert als:

$$typR : R \rightarrow RT$$

Aufgrund der Strukturierung von Typen durch Vererbung ist eine Hilfsfunktion[1] zur Bestimmung aller von einem Komponententyp kt bzw. Relationentyp rt erbenden Typen sinnvoll. Die beiden Funktionen *erbenVonKT* und *erbenVonRT* werden auf der gegenüberliegenden Seite definiert.

[1] Definition 4.12 und Definition 4.13 folgen der Bestimmung der „Dead Successors“ in Leymann und Roller [LR00].

Definition 4.12 (Alle Kindtypen eines Komponententyps)**.** Sei KT die Menge aller Komponententypen, dann ist *erbenVonKT* definiert als:

$$erbenVonKT: KT \to \wp(KT),\ kt \mapsto KT_n$$

Dabei wird KT_n wie folgt bestimmt:

$$KT_0 = \{kt\}$$
$$KT_i = KT_{i-1} \cup \bigcup_{x \in KT_{i-1}} \{y \mid erbtVonKT(y) = x\}, \text{ für } i \geq 1$$

Die Folge von Mengen $(KT_i)_{i \in \mathbb{N}_0}$ ist monoton steigend. Da $KT_i \subseteq KT$ und $|KT| < \infty$, wird $(KT_i)_{i \in \mathbb{N}_0}$ stationär, das heißt

$$\exists n \in \mathbb{N}_0 : KT_0 \subset KT_1 \subset ... \subset KT_n = KT_{n+1} = ...$$

Definition 4.13 (Alle Kindtypen eines Relationentyps)**.** Sei RT die Menge aller Relationentypen, dann ist *erbenVonRT* definiert als:

$$erbenVonRT: RT \to \wp(RT),\ rt \mapsto RT_n$$

Dabei wird RT_n wie folgt bestimmt:

$$RT_0 = \{rt\}$$
$$RT_i = RT_{i-1} \cup \bigcup_{x \in RT_{i-1}} \{y \mid erbtVonRT(y) = x\}, \text{ für } i \geq 1$$

Die Folge von Mengen $(RT_i)_{i \in \mathbb{N}_0}$ ist monoton steigend. Da $RT_i \subseteq RT$ und $|RT| < \infty$, wird $(RT_i)_{i \in \mathbb{N}_0}$ stationär, das heißt

$$\exists n \in \mathbb{N}_0 : RT_0 \subset RT_1 \subset ... \subset RT_n = RT_{n+1} = ...$$

Die Funktionen *typK* und *typR* dürfen keinen Zyklus bilden, das heißt ein Typ darf nicht von sich selbst erben: $\forall kt \in KT: kt \notin erbenVonKT(kt)$ und $\forall rt \in RT: rt \notin erbenVonRT(rt)$

4.2.4 Elemente und ihre Eigenschaften

Als Elemente werden sowohl die Komponenten als auch die Relationen eines ETG verstanden.

Definition 4.14 (Elemente)**.** Sei K die endliche Menge aller Komponenten eines ETGs, sei R die Menge aller Relationen dieses ETGs, dann heißt deren Vereinigung *Elemente*.

$$\mathit{Elemente} = K \cup R$$

Wie im Metamodell in Abbildung 4.2 beschrieben, kann jedem Element eine beliebige Anzahl von Eigenschaften zugeordnet werden. Diese Name-Wert-Paare können verschiedene Arten von Informationen enthalten, insbesondere die Konfiguration oder Laufzeitinformationen des Elements. Die Werte können beliebig lang bzw. komplex sein, das heißt auch ein domänenspezifisches Datenformat nutzen, das jedoch in eine Zeichenkette konvertiert werden muss. Somit ist es möglich, strukturierte Daten als den Wert einer Eigenschaft zu setzen, zum Beispiel eine Konfiguration in XML.

Definition 4.15 (Eigenschaften)**.** Sei *Elemente* die Menge aller Komponenten und Relationen eines ETGs (vgl. Definition 4.14), sei *Zeichen* eine endliche Menge, sei $N = \mathit{Zeichen}^+$ die Menge aller Namen, sei $W = \mathit{Zeichen}^* \cup \{\bot\}$ die Menge aller Werte, dann ist die Funktion *hatEigenschaften* definiert als:

$$\mathit{hatEigenschaften} : \mathit{Elemente} \to \wp(N \times W)$$

Dabei gibt $(n, \bot) \in \mathit{hatEigenschaften}(e)$ an, dass für ein Element e kein Wert für den Name n definiert wurde, im Gegensatz zum leeren Wort, welches einen leeren Wert repräsentiert.

Außerdem gilt, dass auf einem Element kein Name mit zwei unterschiedlichen Werten definiert sein darf, d.h. auf jedem Element e hat jeder Name n einen eindeutigen Wert.

$$\forall e \in \mathit{Elemente}\ \forall (n_1, w_1), (n_2, w_2) \in \mathit{hatEigenschaften}(e):$$
$$n_1 = n_2 \implies w_1 = w_2$$

Definition 4.15 schränkt die verwendeten Namen von Eigenschaften auf einem Element nicht ein, das heißt, für jeden Name kann auf jedem Element ein Wert hinterlegt werden. Aufgrund der verschiedenen Systeme, die auf dem ETG arbeiten, ist es jedoch sinnvoll, Namen zu standardisieren oder durch standardisierte Präfixe zu gruppieren. Zum Beispiel können Typen (vgl. Abschnitt 4.2.3) bestimmte Namen vorgeben, die wichtig für die dem jeweiligen Typ entsprechende Semantik sind. Ein anderes Beispiel sind Systeme, die auf dem ETG arbeiten und Namen definieren, unter denen sie in verschiedenen Elementen Eigenschaften hinterlegen. Dies kann beispielsweise der in Kapitel 5 eingeführte ETG-Crawler sein oder, in einem zukünftigen Anwendungsfall, ein System, welches Monitoring-Informationen im ETG ablegt. Das Metamodell beschränkt also die Namen von Eigenschaften bewusst nicht, um die Flexibilität und Erweiterbarkeit nicht einzuschränken, erlaubt aber Namen global oder auf Ebene der Typen zu standardisieren. Die standardisierten Namen sind bei Typen Teil deren Semantik, welche in einer Beschreibung festgehalten wird, jedoch nicht Teil des in diesem Kapitel beschriebenen Metamodells.

4.2.5 Grundlegende Relationentypen

Bei Relationen kann zwischen drei grundlegenden Typen und damit verbundenen Semantiken unterschieden werden. Sei die Relation r_i zwischen Komponente $k_a = \mathit{von}(r_i)$ und Komponente $k_b = \mathit{nach}(r_i)$ gegeben, dann sind die grundlegenden Typen, von denen alle anderen Typen von Relationen erben, die folgenden drei:
(i) *hostedOn* beschreibt, dass Komponente k_a auf Komponente k_b installiert oder betrieben wird.

(ii) *connectsTo* beschreibt, dass Komponente k_a sich zu Komponente k_b verbindet, zum Beispiel einen Dienst aufruft, eine Datenbank- oder VPN-Verbindung herstellt.
(iii) *dependsOn* beschreibt, dass Komponente k_a von Komponente k_b auf irgendeine Weise abhängt, zum Beispiel dass Komponente k_b zuerst gestartet oder an einem anderen Ort wie Komponente k_a betrieben werden muss.

Definition 4.16 (Grundlegende Relationentypen)**.** Die drei grundlegenden Typen von Relationen, die von keinem anderen Relationentyp erben:

$$\{hostedOn, connectsTo, dependsOn\} \subseteq RT$$

$$erbtVonRT(hostedOn) = \bot$$
$$erbtVonRT(connectsTo) = \bot$$
$$erbtVonRT(dependsOn) = \bot$$

4.2.6 Segmente

Segmente gruppieren einen Teil des ETGs anhand logischer, organisatorischer oder physikalischer Kriterien oder einer Auswahl des Benutzers. Dazu wird angegeben, welche Komponenten und Relationen des gesamten ETGs im entsprechenden Segment enthalten sind. Da Segmente nur Referenzen auf die gesamte Menge der Komponenten und Relationen enthalten, können sie sich überlappen und Änderungen des Typs oder einer Eigenschaft eines Elements werden sofort in allen Segmenten sichtbar.

Definition 4.17 (Segment)**.** Sei K die Menge aller Komponenten eines ETGs, sei R die Menge aller Relationen dieses ETGs, sei j der Index eines Segments, welches dieses im ETG eindeutig identifiziert, dann ist das Segment s_j definiert als Tupel:

$$s_j = (K_j, R_j)$$

Dabei ist K_j die Menge der Komponenten im Segment s_j und R_j die Menge der Relationen im Segment s_j.

$$K_j \subseteq K$$
$$R_j \subseteq R$$

Definition 4.18 (Segmente eines ETGs)**.** Die endliche Menge aller Segmente s_j eines ETGs ist definiert als $S = \{s_0, s_1, ..., s_n\}$.

Auf Segmente können die beiden Funktionen *ausgehendeRelationen* und *eingehendeRelationen* nicht angewendet werden, da diese auf der Menge der Komponenten K und Relationen R des gesamten ETGs definiert sind. Daher werden diese Funktionen auf Segmenten wie folgt definiert:

Definition 4.19 (Ein-/ausgehende Relationen in Segmenten)**.** Sei $s_j = (K_j, R_j)$ ein Segment, dann sind die Funktionen $ausgehendeRelationen_j$ und $eingehendeRelationen_j$, analog zu Definition 4.5, definiert als:

$$ausgehendeRelationen_j : K_j \to \wp(R_j),\ k \mapsto \{r \mid von(r) = k\}$$
$$eingehendeRelationen_j : K_j \to \wp(R_j),\ k \mapsto \{r \mid nach(r) = k\}$$

4.3 Granularität von Enterprise Topologie Graphen

Die ETG-Definition gibt die Granularität der Komponenten, das heißt wie groß der Teil der IT ist, welcher von einer einzelnen Komponente repräsentiert wird, nicht vor. Für jeden ETG kann die Granularität vom Benutzer, in Abhängigkeit von den geplanten Anwendungsfällen, gewählt werden. Ein Datenbanksystem kann zum Beispiel als eine grob-granulare Komponente dargestellt werden, die neben dem DBMS auch noch das Betriebssystem, die virtuelle Maschine und die Hardware repräsentiert. Auf der anderen Seite kann der gleiche Ausschnitt der IT durch eine Vielzahl verschiedener Komponenten repräsentiert werden, indem jede Hardwarekomponente

(Festplatte, Arbeitsspeicher etc.), jedes Betriebssystempaket (Dateisystem, Synchronisation der Zeit etc.) und jede Datei des RDBMS als eigene feingranulare Komponente dargestellt wird. Der im vorhergehenden Beispiel gewählte Mittelweg sind drei Komponenten, jeweils eine für das DBMS, das Betriebssystem und die virtuelle Maschine.

Eine *abgestimmte Granularität* innerhalb eines ETGs ist wichtig, damit verschiedene Benutzer und Systeme, die den ETG nutzen, zusammenarbeiten können. ETG-übergreifend ist eine abgestimmte Granularität sinnvoll, um bestehendes Wissen und Werkzeuge wiederverwenden zu können. Abgestimmt heißt dabei nicht zwangsläufig die gleiche Granularität für alle Anwendungsbestandteile, sondern, dass die Granularität eindeutig definiert und allen Benutzern und Systemen bekannt ist. Bei der Migration in der vorliegenden Arbeit wird beispielsweise ein ESB nicht als eine Komponente dargestellt, sondern jede Route, das heißt wie ein Aufruf verarbeitet wird, als eigene Komponente repräsentiert. Der ESB wird somit in einer feineren Granularität repräsentiert als beispielsweise ein Webserver. Die Typen von Komponenten (vgl. Abschnitt 4.2.3) dokumentieren explizit die abgestimmte Granularität, indem sie definieren, welche Komponenten es gibt und welche nicht. Somit kann der Benutzer durch die Definition (bzw. Auswahl) der Typen die Granularität definieren.

Als Daumenregel hat sich bewährt, dass jede Komponente, die separat installiert, bereitgestellt, gekauft oder verwaltet werden kann, auch als separate Komponente im ETG vorhanden sein sollte. Sinnvoll kann es auch sein, die Typen an ein existierendes, möglicherweise standardisiertes, Typsystem anzulehnen. Für den Prototyp dieser Arbeit wird das verwendete Typsystem in Abschnitt 6.1 beschrieben.

4.4 Topologie Queries

Topologie Queries durchsuchen einen ETG, ausgehend von einer gegebenen Komponente, und bestimmen, welche Komponenten, Relationen oder Eigenschaften den definierten Bedingungen entsprechen. Topologie Queries sind erforderlich, da bei der Suche auf dem ETG der Kontext eine wichtige Rolle spielt, wie im folgenden Beispiel deutlich wird: Um den Hostnamen eines Dienstes zu bestimmen, beispielsweise damit andere Komponenten diesen aufrufen können, muss die Komponente gefunden werden, welche diesen Dienst betreibt. Dazu wird ein Topologie Query ausgeführt, der ausgehend von der Komponente, welche den Dienst repräsentiert, allen ausgehenden Relationen vom Typ *hostedOn* folgt, bis eine Komponente gefunden wird, welche die Eigenschaft `Hostname` definiert. Abbildung 4.3 zeigt exemplarisch, wie drei Relationen vom Typ *hostedOn* traversiert werden müssen, um die Komponente vom Typ *Server* zu finden, der die Eigenschaft *Hostname* definiert. Hierbei ist es wichtig von der Komponente auszugehen, welche den Dienst repräsentiert, und nicht global nach Komponenten mit der Eigenschaft `Hostname` zu suchen, weil diese für den Betrieb einer anderen Komponente verantwortlich sein könnten.

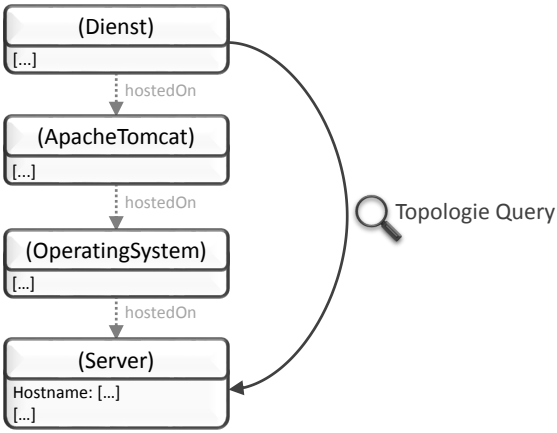

Abbildung 4.3: Darstellung eines exemplarischen Topologie Queries

Ein Topologie Query tq besteht aus einer Startkomponente $k_{start} \in K$, beispielsweise dem Dienst in Abbildung 4.3, und einer Menge von Bedingungen. Das Resultat ist eine Menge von Komponenten $K_{tq} \subseteq K$, die keine, eine oder mehrere Komponenten enthält, auf die alle Bedingungen zutreffen. Die folgenden fünf Arten von Bedingungen werden von Topologie Queries unterstützt, welche nach Komponenten suchen:

(i) Der Abstand der Komponente von k_{start} in traversierten ETG-Relationen, wobei der minimale ($minTiefe \in \mathbb{N}_0$) und/oder maximale ($maxTiefe \in \mathbb{N}_0$) Abstand angegeben werden kann. Falls der Abstand nicht gesetzt wird, ist $minTiefe = 0$ und $maxTiefe = \infty$.

(ii) Die Menge von Komponententypen $KT_{tq} \subseteq KT$, in welcher der Typ der gesuchten Komponenten enthalten sein muss. Falls $KT_{tq} = \emptyset$ sind alle Komponententypen zulässig.

(iii) Die Menge von Relationentypen $RT_{tq} \subseteq RT$, in welcher der jeweilige Typ aller Relationen enthalten sein muss, welche auf dem Weg zur gesuchten Komponente traversiert werden. Falls $RT_{tq} = \emptyset$ sind alle Typen zulässig.

(iv) Ob nur ausgehenden, nur eingehenden oder allen Relationen gefolgt werden soll, wird durch Setzen der Variablen *folgeAusgehendenRelationen* und *folgeEingehendenRelationen* auf *true* oder *false* definiert.

(v) Außerdem können die Eigenschaften der gesuchten Komponenten näher spezifiziert werden. Für eine durch ihren Namen identifizierte Eigenschaft kann verlangt werden, dass sie einen bestimmen Wert hat, vorhanden sein muss oder nicht vorhanden sein darf. Dies wird durch die Funktion $tqBedEigenschaft: N \rightarrow W \cup \{KEIN\_WERT, BELIEBIGER\_WERT\}$ definiert.

Ausgehend von der Komponente k_{start}, wird bei der Ausführung eines Topologie Queries der ETG durchlaufen und die Komponenten zurückgegeben, auf die alle Bedingungen zutreffen. Algorithmus 4.1 zeigt die rekursive Funktion *topologieQuery*, welche Topologie Queries für die Suche nach Komponenten als Tiefensuche [Sch01] realisiert. Die Variablen (*minTiefe*, *maxTiefe*, *folgeAusgehendenRelationen*, *folgeEingehendenRelationen*), die Mengen (KT_{tq}, RT_{tq}) und die Funktionen (*tqBedEigenschaft*) welche die zuvor eingeführten Bedingungen darstellen stehen dabei in Algorithmus 4.1 und 4.2 global zur Verfügung. Zur Ausführung eines Topologie Queries wird die Funktion

Algorithmus 4.1 *topologieQuery*$(k \in K, tiefe \in \mathbb{N}_0)$

```
1:  K_tq ← ∅
2:  if tiefe < maxTiefe then
3:      if folgeAusgehendenRelationen then
4:          for all r ∈ ausgehendeRelationen(k) do
5:              if RT_tq = ∅ ∨ typR(r) ∈ RT_tq then
6:                  K_tq ← K_tq ∪ topologieQuery(nach(r), tiefe + 1)
7:              end if
8:          end for
9:      end if
10:     if folgeEingehendenRelationen then
11:         for all r ∈ eingehendeRelationen(k) do
12:             if RT_tq = ∅ ∨ typR(r) ∈ RT_tq then
13:                 K_tq ← K_tq ∪ topologieQuery(von(r), tiefe + 1)
14:             end if
15:         end for
16:     end if
17: end if
18: if tiefe ≥ minTiefe ∧ pruefeEigenschaften(k) ∧ (KT_tq = ∅ ∨ typK(k) ∈ KT_tq)
    then
19:     K_tq ← K_tq ∪ k
20: end if
21: return K_tq
```

topologieQuery mit der entsprechenden Startkomponente k_{start} und der Tiefe 0 aufgerufen. Die Zeilen 2–17 von Algorithmus 4.1 prüfen dann, ob eine neue Rekursion begonnen werden soll, und Zeilen 18–20, ob alle Bedingungen auf die aktuelle Komponente k zutreffen, so dass diese zur Ergebnismenge K_{tq} hinzugefügt werden kann. Im Detail wird zuerst in Zeile 2 geprüft, ob die maximale Tiefe schon erreicht wurde. Falls nicht, wird für alle von Komponente k ausgehenden (Zeilen 3–9) bzw. eingehenden (Zeilen 10–16) Relationen geprüft, ob deren Typ in der Menge der erlaubten Relationentypen RT_{tq} enthalten ist. Wenn dies der Fall ist, wird die Funktion *topologieQuery* rekursiv aufgerufen (Zeile 6 bzw. Zeile 13). Die aktuelle Komponente k wird zur Ergebnismenge K_{tq} hinzugefügt (Zeile 19), falls alle Bedingungen in Zeile 18

erfüllt sind: Dazu muss die minimale Tiefe erreicht worden sein, müssen die Eigenschaften der Komponente den Bedingungen im Topologie Query entsprechen und muss der Komponententyp von k in der Menge der zulässigen Komponententypen KT_{tq} enthalten sein. Die Eigenschaften der Komponente k werden von der Funktion *pruefeEigenschaften* in Algorithmus 4.2 geprüft.

Algorithmus 4.2 unterscheidet für jede Bedingung, welche für die Eigenschaften definiert wurde (Zeile 1), wird zwischen den drei Fällen (i) kein Wert (Zeilen 2–7), (ii) beliebiger Wert (Zeilen 8–13) und (iii) definierter Wert (Zeilen 14–20). Danach wird in Zeile 3, 9 und 15 die entsprechende Bedingung geprüft. Ist diese Bedingung erfüllt, wird die nächste geprüft, ansonsten *false* zurückgegeben. Wurden alle Bedingungen an die Eigenschaften der gesuchten Komponente erfüllt, wird *true* zurückgegeben.

Algorithmus 4.2 *pruefeEigenschaften*$(k \in K)$

```
 1: for all (key, value) ∈ tqBedEigenschaft do
 2:     if value = KEIN_WERT then
 3:         if (key, ⊥) ∈ hatEigenschaften(k) then
 4:             continue
 5:         else
 6:             return false
 7:         end if
 8:     else if value = BELIEBIGER_WERT then
 9:         if (key, ⊥) ∉ hatEigenschaften(k) then
10:             continue
11:         else
12:             return false
13:         end if
14:     else
15:         if (key, value) ∈ hatEigenschaften(k) then
16:             continue
17:         else
18:             return false
19:         end if
20:     end if
21: end for
22: return true
```

In den vorangegangenen Ausführungen wurde beschrieben, wie Topologie Queries für die Suche von Komponenten genutzt werden können. Die Suche nach Relationen ist analog dazu. Die Suche nach einer Eigenschaft besteht aus einem Topologie Query für Komponenten bzw. Relationen sowie einer Liste von Namen, welche die gesuchten Eigenschaften identifizieren die gefunden werden sollen. Somit ermöglichen es Topologie Queries sowohl Komponenten und Relationen als auch einzelne Eigenschaften von diesen zu bestimmen.

4.5 Operation DeepDive – durch ausgehende Relationen erreichbare Komponenten

Die Operation *deepDive* bestimmt das Segment $s_{deepDive}$, eines ETGs, welches ausgehend von der Menge der gegebenen Komponenten $K_{eingabe} \subset K$ alle Komponenten $K_{deepDive}$ und Relationen $R_{deepDive}$ bestimmt, die zu deren Betrieb nötig sind. Dazu verwendet Algorithmus 4.3 eine Breitensuche, wie beispielsweise von Schöning [Sch01] beschrieben, die alle von den Komponenten in $K_{eingabe}$ durch ausgehende Relationen erreichbaren Komponenten ermittelt. *Rand* ist dabei die Menge der noch zu bearbeitenden Komponenten und $s_{deepDive}$ das Ergebnissegment bestehend aus den beiden Mengen $K_{deepDive}$ und $R_{deepDive}$, welche die entsprechenden Komponenten und Relationen enthalten. In Algorithmus 4.3 wird, solange *Rand* nicht leer ist (Zeile 4), eine Komponente k aus *Rand* (Zeile 5) gewählt und aus *Rand* entfernt (Zeile 6). Dazu wird jede von k ausgehende Relation r (Zeile 7), die nicht bereits im Ergebnissegment enthalten ist (Zeile 8), wie folgt behandelt: (i) Die Zielkomponente $nach(r)$ der Relation r wird zum Rand hinzugefügt (Zeile 9), damit in einem der nächsten Durchläufe deren ausgehenden Relationen untersucht werden. (ii) Die Relation r und ihre Zielkomponente wird dem Ergebnissegment hinzugefügt (Zeilen 12–10). Das Ergebnis ist eine Menge, welche alle Komponenten enthält, die zum Betrieb der übergebenen Komponenten nötig sind.

Algorithmus 4.3 *deepDive*($K_{eingabe} \subseteq K$)

1: $Rand \leftarrow K_{eingabe}$
2: $K_{deepDive} \leftarrow K_{eingabe}$
3: $R_{deepDive} \leftarrow \emptyset$
4: **while** $|Rand| \neq 0$ **do**
5: $k \leftarrow$ erste Komponente aus $Rand$
6: $Rand \leftarrow Rand \setminus \{k\}$
7: **for all** $r \in ausgehendeRelationen(k)$ **do**
8: **if** $nach(r) \notin K_{deepDive}$ **then**
9: $Rand \leftarrow Rand \cup \{nach(r)\}$
10: $K_{deepDive} \leftarrow K_{deepDive} \cup \{nach(r)\}$
11: **end if**
12: $R_{deepDive} \leftarrow R_{deepDive} \cup \{r\}$
13: **end for**
14: **end while**
15: **return** $s_{deepDive} = (K_{deepDive}, R_{deepDive})$

4.6 Identifikation und Isolation der zu migrierenden Komponenten aus dem ETG

Dieser Abschnitt zeigt, wie auf Basis der ETGs und der eingeführten Operationen Schritt 2 („Identifikation, Partitionierung und Isolation der Anwendung aus der Enterprise Topologie“) der AROMA-Methode realisiert werden kann. Der Benutzer wählt dafür eine Menge von Komponenten $K_{gewaehlte} \subset K$ aus (Benutzereingabe für Schritt 2), welche die Anwendungsfunktionalität der zu migrierenden Anwendung(en) repräsentieren. Diese und alle für deren Betrieb nötigen Komponenten werden von dem hier vorgestellten Konzept im ETG identifiziert und isoliert. Das Resultat wird im Segment s_a festgehalten, das die identifizierte und isolierte Anwendung enthält.

Das Konzept für die Identifikation und Isolation der zu migrierenden Komponenten aus dem ETG umfasst drei Schritte: Zuerst wird ein Segment mit den potentiellen Komponenten bestimmt, die für den Betrieb der gewählten Komponenten $K_{gewaehlte}$ höchstens benötigt werden (vgl. Abschnitt 4.6.1). Im zweiten Schritt wird dieses Segment auf die Komponenten reduziert,

die wirklich für den Betrieb der gegebenen Komponenten nötig sind (vgl. Abschnitt 4.6.2). Der dritte und letzte Schritt behandelt gemeinsam genutzte Komponenten, also solche, die sowohl von Komponenten innerhalb als auch außerhalb des Segments s_a benötigt werden (vgl. Abschnitt 4.6.3). Die Schritte 2 und 3 können nicht vollständig automatisiert werden, so dass Rückfragen an den Benutzer nötig sind. Zur Unterstützung werden deshalb Heuristiken bereitgestellt, auf deren Basis der Benutzer dann die Entscheidungen trifft.

4.6.1 Schritt 1: Bestimmung der potentiellen Komponenten

Die potentiellen Komponenten werden durch Algorithmus 4.3 bestimmt. Dieser ermittelt für die Komponenten in $K_{gewaehlte}$ alle durch ausgehende Relationen erreichbare Komponenten. Das Segment $s_a = deepDive(K_{gewaehlte})$ enthält damit alle für den Betrieb der Komponenten in $K_{gewaehlte}$ nötigen Komponenten, möglicherweise aber mehr Komponenten, als migriert werden sollen.

4.6.2 Schritt 2: Reduktion der Komponenten

Nicht alle durch ausgehende Relationen erreichbaren Komponenten sind zwangsläufig für den Betrieb der Komponenten in $K_{gewaehlte}$ nötig. In Bezug auf den Anwendungsfall dieser Arbeit bedeutet dies, dass sie auch nicht migriert werden sollten. Im zweiten Schritt werden daher die im Segment s_a enthaltenen Komponenten genauer geprüft. Dies wird durch Betrachtung der Relationen getan, welche als einzige eine Komponente (und alle Komponenten, die für deren Betrieb nötig sind) an das Segment s_a *bindet*. Die Entscheidung, was mit einer Relation geschieht, fällt der Benutzer, der dadurch bestimmen kann, welche Teile und Abhängigkeiten der Anwendung er migrieren möchte. Dabei muss zwischen den drei grundlegenden Typen von Relationen unterschieden werden, die in Abschnitt 4.2.3 definiert wurden. Die von diesen Typen von Relationen erbenden Typen werden genauso behandelt wie die ihnen entsprechenden grundlegenden Typen.

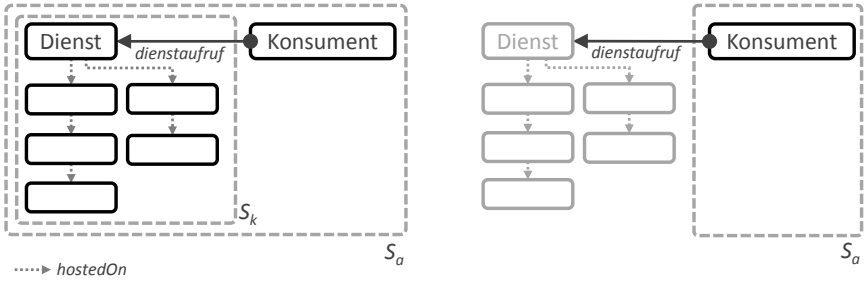

Abbildung 4.4: Beispiel für Reduktion durch Anpassung der Relation

(i) Relationen vom Typ *hostedOn* (und alle Kindtypen) müssen erhalten bleiben, weil sie repräsentieren, dass eine Komponente direkt eine andere Komponente betreibt. Es gibt keine Möglichkeit, auf die dadurch angebundenen Komponenten zu verzichten, ohne die Lauffähigkeit und Funktionalität der Anwendung zu beeinträchtigen.

(ii) Auch Relationen vom Typ *connectsTo* (und alle Kindtypen) müssen zur Sicherstellung der Funktionalität erhalten bleiben. Eine Menge von Komponenten, die nur durch eine Relation r vom Typ *connectsTo* mit den anderen Komponenten in s_a verbunden ist, kann nicht migriert werden, wenn die Relation r entsprechend angepasst wird. Abbildung 4.4 zeigt dies, von links nach rechts, für einen Dienst und eine Relation vom Typ $dienstaufruf \in erbenVonRT(connectsTo)$. Falls der Benutzer dies wünscht, kann mit Algorithmus 4.3 das Segment s_k bestimmt werden, das alle Komponenten enthält, die für den Betrieb der Komponente $k = nach(r)$ nötig sind. Diese werden dann aus dem Segment s_a entfernt. Dabei muss in der Ursprungs- und Zielumgebung sichergestellt werden, dass eine Netzwerkverbindung möglich ist und die entsprechenden Sicherheitsvorgaben einen Dienstaufruf zulassen. Dadurch kann verhindert werden, dass der gesamte Dienst mit allen zu seinem Betrieb nötigen Komponenten migriert werden muss. Daher spielen Relationen vom Typ *connectsTo* eine wichtige Rolle bei

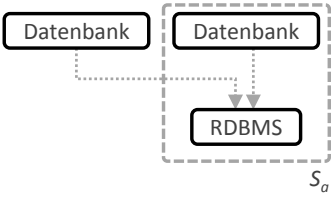

Abbildung 4.5: DBMS als gemeinsam genutzte Komponente

der Reduktion der Komponenten, verlangen aber eine manuelle Entscheidung darüber, welche Komponenten als Teil der zu migrierenden Anwendung angesehen werden sollen und welche nicht.

(iii) Relationen vom Typ *dependsOn* (und alle Kindtypen) sind teilweise technisch nicht oder nicht mehr relevant, wenn die Komponenten migriert werden. Sie repräsentieren üblicherweise eine logische Abhängigkeit, aber keine Verbindung oder Installation wie Relationen vom Typ *connectsTo* und *hostedOn*. Manche Semantiken, zum Beispiel „starte a nach b“, können nach der Migration möglicherweise nicht mehr dargestellt werden. Falls eine Komponente k ausschließlich über eine Relation vom Typ *dependsOn* mit den anderen Komponenten im Segment s_a verbunden ist, sollte sie manuell geprüft werden. Dazu wird mit Algorithmus 4.3 das Segment s_k bestimmt, welches die Komponente k und alle für ihren Betrieb nötigen Komponenten enthält. Wenn sich herausstellt, dass diese Abhängigkeit nicht mehr nötig ist oder zutrifft, kann das gesamte Segment s_k aus dem Segment s_a entfernt werden. Stattdessen kann es sinnvoll sein, den Aspekt als Policy der zu migrierenden Anwendung zu externalisieren. Derartige Policies müssen dann bei der Bereitstellung der Anwendung berücksichtigt werden. Dies muss vom entsprechenden Experten entschieden werden.

Dieser Schritt hat die Anzahl der Komponenten, die für den Betrieb der übergebenen Komponenten nötig sind, reduziert. Dies verhindert, dass für den Betrieb nicht benötigte Komponenten migriert werden.

4.6.3 Schritt 3: Behandlung gemeinsam genutzter Komponenten

Darüber hinaus können im Segment s_a Komponenten enthalten sein, die auch von Komponenten außerhalb des Segments s_a genutzt werden. Wie in Abbildung 4.5 veranschaulicht, kann dies zum Beispiel bei einem RDBMS der Fall sein, wenn eine Datenbank eine Komponente in s_a ist und eine, welche auf demselben RDBMS betrieben wird, nicht. Algorithmus 4.4 identifiziert Komponenten, die sowohl von Komponenten in als auch von Komponenten außerhalb des Segments s_a verwendet werden. Dies ist der Fall, wenn in Zeile 3 Komponente k im gesamten ETG mehr eingehende Relationen hat als im Segment s_a (vgl. Definition 4.5 und Definition 4.19).

Algorithmus 4.4 *gemeinsamGenutzteKomponenten*$(K_a \subseteq K)$

```
1: K_gemeinsamGenutzte ← ∅
2: for all k ∈ K_a do
3:     if |eingehendeRelationen(k)| > |eingehendeRelationen_a(k)| then
4:         K_gemeinsamGenutzte ← K_gemeinsamGenutzte ∪ {k}
5:     end if
6: end for
7: return K_gemeinsamGenutzte
```

Unabhängig davon, ob diese Relationen vom Typ *hostedOn* oder *connectsTo* ist, gibt es drei Möglichkeiten, eine gemeinsam genutzte Komponente $k_g \in$ *gemeinsamGenutzteKomponenten*(K_a) zu behandeln:

(i) *Aufteilen.* Oft sind gemeinsam genutzte Komponenten Middleware, wie zum Beispiel ein DBMS oder Application Server. Diese haben oft mehrere eingehende Relationen vom Typ *hostedOn*, die von verschiedenen Komponenten ausgehen, zum Beispiel Datenbanken oder Anwendungen. Die darauf basierenden Komponenten haben teilweise keine Abhängigkeiten untereinander, sondern werden aus Gründen der Effizienz auf einer gemeinsamen Komponente betrieben. In diesem Fall kann die gemeinsam genutzte Komponente k_g aufgeteilt werden, indem sie kopiert wird, wie in Abbildung 4.6 links dargestellt: Die Komponente k_g bleibt also in der Ursprungsumgebung unverändert, während Komponente $k_g{}'$ als Kopie von k_g im Segment s_a an-

gelegt wird. Die eingehenden Relationen von Komponenten aus s_a werden dann auf $k_g{}'$ *umgeleitet*, formal ausgedrückt:

$$\forall r \in eingehendeRelationen_a(k_g) \colon nach(r) \leftarrow k_g{}'$$

(ii) *Synchronisieren.* Falls die gemeinsam genutzte Komponente hingegen Daten hält, die von Komponenten außerhalb und innerhalb des Segments s_a geteilt werden, beispielsweise in einer gemeinsam verwendeten Datenbanktabelle, ist die zuvor beschriebene Aufteilung nicht möglich. In diesem Fall ist, neben der Aufteilung der gemeinsam genutzten Komponente, auch die Synchronisation der gemeinsam genutzten Daten zwischen Komponente k_g und $k_g{}'$ nötig (siehe Mitte Abbildung 4.6). Dies muss von der entsprechenden Middleware unterstützt oder manuell implementiert werden und erfordert eine Netzwerkverbindung zwischen der Ursprungs- und Zielumgebung.

(iii) *Weiterhin gemeinsam nutzen.* Falls die Synchronisation der gemeinsam genutzten Komponenten nicht gewünscht, sinnvoll oder realisierbar ist, kann Komponente k_g auch weiterhin gemeinsam genutzt werden, wie rechts in Abbildung 4.6 in zwei Ausprägungen dargestellt. Dies verlangt, in Abhängigkeit davon, ob die gemeinsam genutzte Komponente in der Ursprungs- oder Zielumgebung betrieben werden soll, die manuelle Anpassung der darauf zugreifenden Komponenten. Falls k_g in der Ursprungsumgebung bleibt, muss die Relation von den Komponenten im Segment s_a durch die Einführung einer externen Abhängigkeit sichergestellt werden.

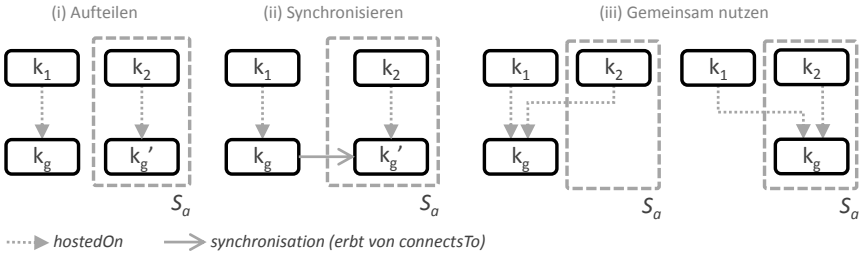

Abbildung 4.6: Übersicht der Lösungsmöglichkeiten für gemeinsam genutzte Komponenten

4.6.4 Erweiterbarkeit

Bei der Einführung neuer grundlegender Relationentypen muss deren Semantik in Bezug auf die Fallunterscheidung in Schritt 2 (Abschnitt 4.6.2) untersucht werden. Idealerweise kann der neue Relationentyp auf einen der drei in Abschnitt 4.6.2 diskutierten Fälle abgebildet werden oder es ist eine allgemeingültige Entscheidung möglich, wie beispielsweise bei Relationen vom Typ *hostedOn*. Falls Relationen dieses Typs für jede Anwendung neu betrachtet werden müssen, kann möglicherweise zumindest eine Heuristik gefunden werden, die den Benutzer bei seiner Entscheidung unterstützt, wie beispielsweise zuvor für Relationen vom Type *connectsTo* diskutiert. Falls zum Beispiel der Relationentyp *polls*[1] unterstützt werden soll, könnte dies auf den Fall (ii) in Schritt 2 abgebildet werden. Wie beim Relationentyp *connectsTo* muss dann der Benutzer entscheiden, ob er den Dienst, der regelmäßig aufgerufen wird, migrieren will oder nicht. Die Schritte 1 und 3 sind unabhängig von den im ETG verwendeten Relationentypen.

4.6.5 Diskussion

Die Bestimmung des Segments s_a, das eine Menge von durch den Benutzer gegebenen Komponenten $K_{gewaehlte}$ betreibt, kann in verschiedenen Anwendungsfällen genutzt werden. Bei der Migration von Anwendungen spielt die Identifikation der zu migrierenden Komponenten eine wichtige Rolle. Besonders gemeinsam genutzte Komponenten sind dabei eine Herausforderung.

Dieser Abschnitt hat gezeigt, wie die Bestimmung der potentiellen Komponenten in Schritt 1 automatisiert werden kann. Die Reduktion der Komponenten auf die wirklich nötigen (Schritt 2) und die Behandlung gemeinsam genutzter Komponenten (Schritt 3) müssen manuell durchgeführt werden, da dazu Expertenwissen über die Anwendung, Komponenten und das Ziel der Migration nötig ist. Zur Unterstützung der Experten wurden verschiedene Entscheidungsmöglichkeiten und Heuristiken vorgestellt und diskutiert.

[1] Eine Komponente erfragt regelmäßig bei einer anderen Komponente Informationen.

4.7 Diskussion und Zusammenfassung

In diesem Kapitel wurde mit dem Enterprise Topologie Graph (ETG) ein Metamodell für die Repräsentation der IT eingeführt und Operationen auf diesem definiert. Ein ETG ist ein (i) technisch detailliertes, (ii) alle Arten von Anwendungsbestandteilen umfassendes, (iii) erweiterbares, (iv) für verschiedene Anwendungsfälle zugängliches, (v) anwendungsübergreifendes und (vi) formales Instanzmodell der gesamten IT einer Organisation.

ETGs wurden bereits in einer Vielzahl von anderen Arbeiten und Anwendungsfällen verwendet, was ihren praktischen Nutzen, ihre Anwendbarkeit, Erweiterbarkeit und Zugänglichkeit belegt. Das übergreifende Thema ist dabei die Optimierung der IT, welche als ETG repräsentiert wird. Mithilfe der in Binz et al. [BLNS12] vorgestellten Strategien lässt sich der ETG analysieren, um zu bewerten, welche Schritte im Folgenden zur Optimierung einzuleiten sind. Nowak et al. [NBLU13] nutzen ETGs, um Geschäftsprozesse hinsichtlich ihrer ökologischen Auswirkungen zu analysieren und zu optimieren. Dies zeigt auch, dass die Repräsentation von Geschäftsprozessen in ETGs wichtig ist. Binz et al. [BFL+12] zeigen, wie ETGs genutzt werden können, um mehrfach vorhandene Komponenten zu identifizieren und wo sinnvoll zusammenzuführen. Damit werden – neben der Verbesserung der ökologischen Gesichtspunkte der IT – auch die Kosten für die Verwaltung, Lizenzen und Hardware der Komponenten reduziert. Breitenbücher et al. [BBKL13b, BBL14] beschreiben einen Ansatz, der bestehende Anwendungen, die durch einen ETG repräsentiert werden, zur Laufzeit adaptiert. Binz et al. [BLNS12] zeigen, wie aus einem ETG automatisiert eine Enterprise Architektur erstellt werden kann. Die vorliegende Arbeit nutzt ETGs als Grundlage für die Durchführung einer Anwendungsmigration von einer Ursprungsumgebung, die als ETG vorliegt, in eine Zielumgebung.

Jedoch existiert normalerweise kein ETG der Ursprungsumgebung oder ein vergleichbares IT-Instanzmodell, welches die zu migrierenden Komponenten und deren Abhängigkeiten enthält. Daher betrachtet das nachfolgende Kapitel 5 im Detail die automatisierte Erstellung von ETGs.

KAPITEL 5

CRAWLING VON ENTERPRISE TOPOLOGIE GRAPHEN

Das Crawling eines IT-Instanzmodells ist Thema dieses Kapitels und Beitrag 3 der vorliegenden Arbeit. Das vorhergehende Kapitel 4 hat mit Enterprise Topologie Graphen (ETGs) ein Metamodell zur Repräsentation und Verarbeitung von Enterprise Topologien eingeführt. Ein ETG oder ein ähnliches Instanzmodell existiert normalerweise nicht, da die IT-Komponenten meistens manuell verwaltet werden, sich regelmäßig weiterentwickeln und keine Dokumentation geführt wird. Meist sind nicht einmal die Abhängigkeiten zwischen verschiedenen Komponenten vollständig bekannt [KBK00]. Dieser fehlende Einblick kann dazu führen, dass Veränderungen an der IT langsam und teuer sind oder sogar folgenreiche Fehlentscheidungen getroffen werden [BBK+14b]. Aufgrund der wachsenden Bedeutung der IT für die Wettbewerbsfähigkeit von Firmen ist ein technisch detailliertes, vollständiges, korrektes und aktuelles Verständnis der IT, insbesondere der Abhängigkeiten, von fundamentaler Wichtigkeit.

Eine Enterprise Topologie kann entweder manuell modelliert oder automatisiert generiert werden. Wie in Abschnitt 2.6 diskutiert, fokussiert sich die vorliegende Arbeit auf das automatisierte Erstellen der Enterprise Topologie, da die manuelle Modellierung zeitaufwändig, fehleranfällig und teuer ist [Ens99, MDW+99, BKK01, FAB+10]. Deshalb ist auch für die Umsetzung der AROMA-Methode nur ein automatisiertes Crawling der Enterprise Topologie sinnvoll (vgl. Anforderung A-3 in Abschnitt 3.1). Für die Umsetzung der AROMA-Methode ist der ETG das zentrale Datenmodell und dessen automatisierte Erstellung somit ein wichtiger Beitrag.

Die Betrachtung der verwandten Arbeiten in Abschnitt 2.6 hat gezeigt, dass es bisher keine Lösung gibt, welche eine komplette Enterprise Topologie erstellen kann. Existierende Lösungen sind entweder auf bestimmte Komponenten beschränkt (siehe Abschnitt 2.6.5), bestimmen die Struktur oder Abhängigkeiten nur auf Ebene der Dienste oder des Netzwerks (siehe Abschnitt 2.6.3), erkennen lediglich bestimmte, vordefinierte Anwendungen (siehe Abschnitt 2.6.4) oder sind zu abstrakt und nicht auf technische Details fokussiert (siehe Abschnitt 2.6.1 und Abschnitt 2.6.2). In Bezug auf die Arten von Anwendungsbestandteilen (Abschnitt 2.2) ist festzustellen, dass Geschäftsprozesse und deren Dienstaufrufe durch Enterprise Service Busse nicht von den existierenden Arbeiten adressiert werden [LaC07]. Es zeigt sich aber auch, dass es eine Vielzahl von bewährten Lösungen gibt, die einen Teil der Enterprise Topologie abdecken, und dass somit eine Integration verschiedener Informationsquellen und Werkzeuge sinnvoll ist.

Die Fragestellung dieses Kapitels ist somit: Wie kann ein ETG automatisiert erstellt werden? Dabei ist es wichtig, dass auch Geschäftsprozesse abgebildet werden können und die Integration bestehender Informationsquellen möglich ist. Neben dem selektiven Crawling eines Teils des ETGs soll auch die Aktualisierung bestehender ETGs möglich sein.

Ausgehend von den Anforderungen an die Crawler-Methode diskutiert Abschnitt 5.1 deren Konzept, Architektur, Rollen, Algorithmen, die Integration bestehender Informationsquellen und Werkzeuge sowie die Sicherstellung der Qualität des ETGs. Crawler-Plugins stellen den Erweiterungsmechanismus der Crawler-Methode dar. Daher schildert Abschnitt 5.2 detailliert die

Arbeitsweise ausgewählter Crawler-Plugins und Abschnitt 5.3 widmet sich deren Entwicklung und Test. Vor der Diskussion und Zusammenfassung in Abschnitt 5.5 wird die vorgestellte Lösung zum Crawling von ETGs in Abschnitt 5.4 evaluiert und validiert.

5.1 Crawler-Methode

Basierend auf den Anforderungen an die Crawler-Methode, welche in Abschnitt 5.1.1 vorgestellt werden, definiert dieser Abschnitt das Konzept (Abschnitt 5.1.2), die Architektur und Komponenten (Abschnitt 5.1.3), die beteiligten Rollen (Abschnitt 5.1.4), die Ablaufsteuerung (Abschnitt 5.1.5) und die Qualitätssicherung durch Deduplizierung (Abschnitt 5.1.6) der Methode zum Crawling von ETGs.

5.1.1 Anforderungen an die Crawler-Methode

Die Anforderungen an die Crawler-Methode resultieren aus der AROMA-Methode (vgl. Kapitel 3), Literatur und den Erfahrungen mit ETGs in verschiedenen Anwendungsfällen (vgl. Abschnitt 4.7). Die folgenden fünf Anforderungen an die Crawler-Methode wurden identifiziert:

Anforderung 1 (CA-1): ETG-Qualität. Die Qualität ist die erste und wichtigste Anforderung, die sich aus Schritt 1 der AROMA-Methode ableitet und von der auch die Umsetzbarkeit der anderen Anwendungsfälle direkt abhängt. In der vorliegenden Arbeit basiert die Definition von Qualität auf ausgewählten Arbeiten von Wang und String [WS96], Batini und Scannapieco [BS06], Olson und Delen [OD08] sowie Farwick et al. [FAB+11b]. Wang und String [WS96] befragten Nutzer von Daten in Firmen und staatlichen Organisationen und teilen deren Anforderungen an Datenqualität in vier Kategorien ein: Korrektheit (engl. *accuracy*), Relevanz (engl. *relevancy*), Repräsentation (engl. *representation*) und Zugänglichkeit (engl. *accessibility*). Die Arbeit von Batini und Scannapieco [BS06] beleuchtet Datenqualität im Allgemeinen. Olson und Delen [OD08] definieren Kriterien für die Bewertung von Data-Mining-Techniken. Farwick et al. [FAB+11b] führten eine

Befragung von Praktikern durch, um die Anforderungen zu ermitteln, welche diese an die automatisierte Dokumentation der Enterprise Architektur stellen. Die vorliegende Arbeit unterscheidet davon ausgehend zwischen vier Qualitätskriterien eines ETGs:
(i) *Aktualität*. Der ETG repräsentiert die entsprechende IT so, wie sie zum aktuellen Zeitpunkt ist [WS96, BS06, FAB+11b].
(ii) *Vollständigkeit*. Alle Komponenten der entsprechenden IT und ihre Relationen sind im ETG enthalten [BS06, OD08, FAB+11b].
(iii) *Korrektheit*. Der ETG ist richtig und konsistent, enthält also auch nicht mehr Komponenten als in der IT vorhanden [WS96, BS06, OD08, FAB+11b].
(iv) *Richtige Granularität*. Die Granularität, welche im Rahmen der Definition von ETG-Komponenten (Abschnitt 4.3) bereits diskutiert wurde, muss passend für den entsprechenden Anwendungsfall sein [FAB+11b].
Die Vollständigkeit und Korrektheit eines ETGs muss im Kontext der gewählten Granularität bewertet werden, da beispielsweise der ETG bei einer groben Granularität mit weniger Komponenten vollständig ist als bei einer feineren Granularität. Die Korrektheit fordert gleichzeitig, dass die Abstraktion der grobgranularen Komponenten korrekt und konsistent ist und somit keine für die gewählte Granularität wichtige Information fehlt.

Anforderung 2 (CA-2): Erweiterbarkeit. Aufgrund der Heterogenität der Komponenten und Relationen der IT darf die Crawler-Methode die Art der existierenden und zukünftigen Komponenten (z.B. Software- und Hardware-Produkte), deren Konfiguration und Arten von Relationen nicht einschränken. Dies erfordert auch, dass die Typen von Komponenten und Relationen nicht abschließend definiert oder anderweitig eingeschränkt werden. Gleiches gilt für den Weg, auf dem die Informationen für den ETG extrahiert werden. Es ist somit erforderlich, von einer *offenen Welt* auszugehen (engl. *open-world assumption*), in der über die nicht bekannten Zusammenhänge und Fakten keine abschließenden Aussagen getroffen werden können. Dies verlangt, dass die Methode zur Erstellung von ETGs erweiterbar ist.

Anforderung 3 (CA-3): Integration bestehender Werkzeuge. Es muss möglich sein, bestehende Informationsquellen und Werkzeuge in die Crawler-Methode zu integrieren [FAB+11b, HMR12]. Dies ist auch das Ergebnis der

Untersuchung der verwandten Arbeiten in Abschnitt 2.6, die gezeigt hat, dass Werkzeuge existieren, die bereits bestimmte Aspekte der Enterprise Topologie abdecken.

Anforderung 4 (CA-4): Aktualisierung von ETGs. Da sich die IT ständig ändert, muss es möglich sein, diese Änderungen im ETG effizient nachzupflegen, um stets auf aktuellen Daten zu arbeiten [FAB+11b, HMR12]. Außerdem ist die Aktualisierung erforderlich, um das Qualitätskriterium „Aktualität“ (vgl. Anforderung CA-1) effizient erreichen zu können. Die Relevanz der Aktualität für die Anwendungsmigration diskutiert Abschnitt 3.2.1.

Anforderung 5 (CA-5): Einfluss auf Produktionssysteme minimieren. Zum Crawling des ETGs muss auf die Produktionssysteme zugegriffen werden, was zwangsläufig auch Ressourcen, insbesondere Rechenzeit, Speicher und Netzwerkkapazität dieser Systeme erfordert. Damit der Crawler in der Praxis anwendbar ist, muss deshalb der negative Einfluss auf Produktivsysteme minimiert werden [JPRD10, HMR12]. Dazu gehört auch, dass keine Veränderungen der IT-Komponenten nötig sind, zum Beispiel die Installation von Agenten auf den Systemen oder die Anpassung der Konfiguration einer Komponente, damit diese von sich aus Daten an den Crawler sendet.

5.1.2 Konzept

Das grundlegende Konzept ist ein iterativ arbeitender Crawler, der verschiedene *Crawler-Plugins* nutzt, um ETGs zu erstellen. Aufgrund der Heterogenität der IT ist der Crawler grundsätzlich Plugin-basiert, das heißt, die Logik, um Informationen über verschiedene Typen von Komponenten und Relationen zu extrahieren, wird über typspezifische Crawler-Plugins bereitgestellt. Plugins können *alles* tun, um Informationen über Komponenten und deren Relationen herauszufinden. Jedoch müssen die Plugins diese Informationen selbst von der jeweiligen Komponente *extrahieren* (engl. *pull*), weil keine Komponente und kein IT-Verwaltungs-System vom Crawler wissen sollte, indem es Informationen zum Crawler *sendet* (engl. *push*). Damit Komponenten Status- oder Änderungsinformationen an den Crawler schicken, wäre ein deutlicher Eingriff in die bestehende IT nötig, was gegen die Anforderung

CA-5 („Einfluss auf Produktionssysteme minimieren“) verstoßen und die Erweiterbarkeit (CA-2) einschränken würde. Durch diesen erweiterbaren Ansatz kann der Crawler grundsätzlich alle Arten von Protokollen (HTTP, SSH, SCP, SNMP, JMX etc.) und Datenformaten (XML in verschiedenen Schemata, Datenbanken, Text, Property Files, Log-Dateien, Ausgaben auf der Konsole etc.) unterstützen.

Das gewählte Konzept geht überwiegend von *oben nach unten* vor, das heißt, der Crawler startet *oben*, von der Anwendungsfunktionalität aus, und geht von dort nach *unten* in Richtung Infrastruktur und Netzwerk vor. Dieses Vorgehen ist sinnvoll, weil ausgehend von der Anwendungsfunktionalität die Standardisierung und daher die Austauschbarkeit der Komponenten stetig zunimmt. Die Logik, welche durch einen Dienst oder Prozess implementiert wird, ist normalerweise deutlich spezieller als die Middleware und Infrastruktur, die anhand bekannter Parameter konfiguriert werden. Um die Migration dieser Anwendungsfunktionalität zusammen mit allen für deren Betrieb nötigen Komponenten geht es im Anwendungsfall dieser Arbeit. Daher ist oft auch nur eine Komponente mit Anwendungsfunktionalität als „Einstiegspunkt“ bekannt. Das Vorgehen von oben nach unten ist jedoch keine harte Vorgabe, wo sinnvoll können Crawler-Plugins auch die umgekehrte Richtung gehen, insbesondere um Geschwisterkomponenten zu ermitteln.

Im Folgenden wird das Konzept anhand eines exemplarischen Crawler-Laufes mit fünf Iterationen, wie in Abbildung 5.1 dargestellt, verdeutlicht: Hierbei wird angenommen, dass der Benutzer oder ein benutzendes System vor der ersten Iteration eine Einstiegskomponente definiert, von der ausgehend gecrawlt werden soll. Die Einstiegskomponente ist zum Beispiel die Schnittstelle oder der Endpunkt der Anwendung, über die der Benutzer oder andere Systeme mit der Anwendung interagieren. In diesem Beispiel ist dies eine Komponente vom Typ *Application*, die als Eigenschaft die URL der Anwendung enthält. Auf diesem ETG wird der Crawler dann gestartet und in jeder der nachfolgenden Iterationen werden ein oder mehrere Crawler-Plugins ausgeführt. Die in der jeweiligen Iteration ausgeführten Plugins werden in Abbildung 5.1 jeweils unten aufgelistet. Basierend auf den Informationen, die ein Plugin ermittelt, kann es den Typ der Komponente oder der

Abbildung 5.1: Wachsender ETG während eines Crawler-Laufes

Relation ändern, neue Komponenten und Relationen hinzufügen oder deren Eigenschaften ändern. Das in Iteration 1 ausgeführte Plugin „Application“ bestimmt anhand der HTTP-Header, dass es sich um eine Java-Webanwendung handelt. Deshalb ändert das Plugin den Typ der obersten Komponente in *ApplicationJava* und erstellt eine Komponente vom Typ *Webserver* mit dessen URL, ohne weitere Informationen zu extrahieren. Darüber hinaus wird der Webserver mit der Anwendung durch eine Relation vom Typ *deployedOn* verbunden. Durch die Nutzung von verschiedenen Typen von Relationen und das Hinzufügen von Eigenschaften zu Relationen (was in diesem Beispiel nicht gezeigt wird) ist es möglich, eine sehr genaue und feingranulare Sicht der IT zu repräsentieren. In der nachfolgenden Iteration 2 wird der Typ des Webservers bestimmt (*ApacheTomcat*) und deshalb in Iteration 3 das entsprechende Plugin „ApacheTomcatJMX“ ausgeführt, welches zum Beispiel die exakte Version `7.0.42` bestimmt. Als zweites Beispiel soll die Komponente Betriebssystem betrachtet werden, deren Typ sich von *OperatingSystem* in Iteration 2, nach *Linux* in Iteration 3 und *Ubuntu* in Iteration 4 ändert. An diesem Beispiel ist gut zu sehen, wie verschiedene Plugins sowohl den Typ ändern als auch mehr und detailliertere Eigenschaften ermitteln können.

Dies veranschaulicht, dass Crawler-Plugins zusammenarbeiten können, indem Informationen, die von einem Plugin geschrieben wurden, von einem anderen genutzt, verfeinert oder erweitert werden. Dabei sind die verschiedenen Plugins nur über den ETG miteinander gekoppelt und basieren somit auf den global definierten Typen und Eigenschaften der Komponenten und Relationen im ETG (vgl. Abschnitt 4.2.4). Dadurch weiß ein Crawler-Plugin, unter welchem Namen es welche extrahierten Informationen als Eigenschaft ablegen muss, und andere Plugins wissen, was der Wert einer Eigenschaft bedeutet, wenn sie eine Komponente dieses Typs verarbeiten. In welcher Reihenfolge und auf welchen Komponenten Plugins ausgeführt werden, bestimmt die Ablaufsteuerung, die in Abschnitt 5.1.5 vorgestellt wird.

5.1.3 Crawler-Architektur

Dieser Abschnitt beschreibt die Architekturkomponenten einer Crawler-Umgebung und deren Beziehungen. Auf der rechten Seite von Abbildung 5.2 wird die IT gezeigt, von welcher der Crawler einen ETG erstellen soll.

In der „Crawler-Verwaltung“ werden die verschiedenen Crawler-Läufe, basierend auf den für den jeweiligen Lauf ausgewählten und gegebenenfalls konfigurierten Crawler-Plugins, verwaltet. Das Crawling wird von der „Ablaufsteuerung“ durchgeführt, die entscheidet, welche Plugins wann und in welcher Reihenfolge auf welchen ETG-Komponenten ausgeführt werden. Die „Plugin-Verwaltung“ ermöglicht die Integration der Crawler-Plugins in den Crawler. Plugins registrieren sich mit der Liste der Typen, die sie unterstützen, bei der Plugin-Verwaltung, welche diese der Ablaufsteuerung zur Ausführung bereitstellt. Wie auf der rechten Seite von Abbildung 5.2 dargestellt, können Crawler-Plugins existierende Werkzeuge nutzen.

Ziel der Plugin-Verwaltung ist es, möglichst viele Funktionalitäten, die mehrere Crawler-Plugins benötigen, zentral anzubieten. Dazu stellt diese „wiederverwendbare Dienste“ zur Verfügung, beispielsweise das Suchen im ETG, die Verwaltung von Artefakten, das Validieren und Transformieren von Daten oder das Herstellen von Verbindungen (SSH, FTP, ...).

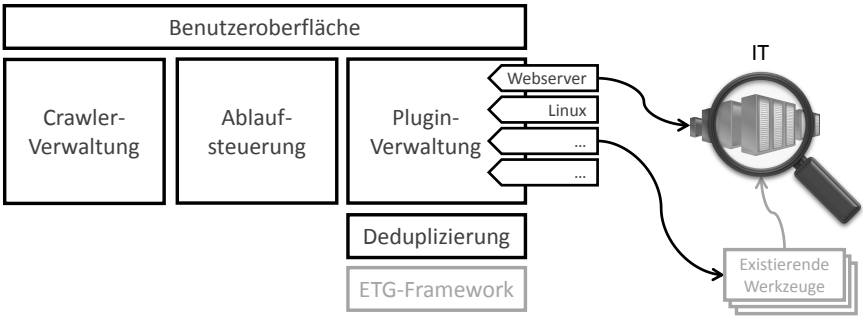

Abbildung 5.2: Architektur einer Crawler-Umgebung

Auf die Verwaltung von Artefakten wird im Folgenden näher eingegangen: Artefakte sind Dateien und größere Mengen an Daten, die Crawler-Plugins von der Zielumgebung extrahieren oder die, auf Nachfrage eines Plugins, vom Benutzer bereitgestellt werden. Da möglicherweise mehrere Plugins das gleiche Artefakt verarbeiten, werden Artefakte zwischengespeichert. Auch vom Benutzer bereitgestellte Artefakte sollten nicht mehrmals angefordert werden, weil dies die Kosten und die Laufzeit eines Crawler-Laufes unnötig erhöhen würde. Artefakte werden von Crawler-Plugins angefragt, die auch entscheiden, ob das Alter des Artefaktes akzeptabel ist und ob es neu geladen oder zumindest überprüft werden muss.

Die Architekturkomponente „ETG-Framework“ wird in Abschnitt 7.2 im Detail behandelt. Dieses stellt die ETGs für den Crawler bereit und ermöglicht den Crawler-Plugins unter anderem Komponenten und Relationen anzulegen, zu verändern und zu löschen. Für Crawler-Plugins wird der Zugang zum ETG durch die Architekturkomponente „Deduplizierung“ abgeschirmt, auf die im Detail in Abschnitt 5.1.6 eingegangen wird. Diese kontrolliert alle Operationen, die von Plugins auf dem ETG ausgeführt werden sollen, und stellt die Qualität der Daten im ETG sicher.

Die „Benutzeroberfläche“ gibt dem Benutzer Zugriff auf die verschiedenen Architekturkomponenten. Die Benutzeroberfläche der Crawler-Verwaltung erlaubt es, Crawler-Läufe zu konfigurieren, zu steuern und zu überwachen. Die Benutzeroberfläche der Ablaufsteuerung ermöglicht es Crawler-Plugins Rückfragen an den Benutzer zu stellen, beispielsweise um nach fehlenden Zugangsdaten zu fragen. Die Benutzeroberfläche der Plugin-Verwaltung wird verwendet um Plugins zu verwalten und zu konfigurieren.

Die in Abbildung 5.2 gezeigte Architektur trennt die Architekturkomponenten nach ihren Verantwortlichkeiten und schafft klare Schnittstellen. Dies erlaubt auch einzelne Komponenten separat zu entwickeln und auszutauschen. Auf die Entwicklung von Crawler-Plugins, dem Erweiterungsmechanismus des Crawlers, wird in Abschnitt 5.3 eingegangen.

5.1.4 Rollen

Für die Umsetzung der Crawler-Methode müssen zwei Rollen benannt werden, welche durch den Crawler nicht automatisierbare Aufgaben übernehmen. Eine Rolle, die den resultierenden ETG nutzt bzw. dessen Erstellung beauftragt, wird hier nicht definiert, da dies vom Anwendungsfall abhängt.

Der *Crawler-Administrator* ist dafür verantwortlich, Crawler-Läufe zu konfigurieren und zu steuern (starten, abbrechen, pausieren) sowie die dafür nötigen Crawler-Plugins zu beschaffen. Die damit betraute Person bearbeitet Rückfragen von Crawler-Plugins (siehe Abschnitt 5.1.5.6) selbst oder ermittelt den entsprechenden Ansprechpartner. Weitere Aufgaben sind damit verbundene organisatorische Angelegenheiten, wie beispielsweise die Gewährung von Zugriffsrechten.

Der *Crawler-Plugin-Entwickler* entwickelt und testet Crawler-Plugins, welche die entsprechenden typspezifischen Informationen für den ETG extrahieren. Die damit betraute Person ist typischerweise Experte für die Komponenten, für die das Plugin entwickelt werden soll und kann somit definieren, welche Informationen relevant sind und wie diese ermittelt werden können.

5.1.5 Ablaufsteuerung

Die Ablaufsteuerung (engl. *scheduler*) bestimmt, in welcher Iteration eines Crawler-Laufes auf welcher ETG-Komponente welches Plugin ausgeführt wird. Um dies zu beschreiben, wird in Abschnitt 5.1.5.1 eine Reihe von grundlegenden Definitionen eingeführt. In Abschnitt 5.1.5.2 wird darauf basierend vorgestellt, wie der gesamte Crawler-Lauf und eine Iteration davon abläuft. Davon ausgehend zeigt Abschnitt 5.1.5.3, wie das Crawler-Plugin bestimmt wird, welches auf einer Komponente in einer Iteration ausgeführt werden soll. Abschnitt 5.1.5.4 erweitert die Ablaufsteuerung, um das „selektive Crawling“ eines ETGs zu ermöglichen. Ausgehend davon wird in Abschnitt 5.1.5.5 diskutiert, wie ein bestehender ETG aktualisiert werden kann, und in Abschnitt 5.1.5.6, wie Rückfragen von Crawler-Plugins an den Benutzer realisiert werden.

5.1.5.1 Definitionen

Dieser Abschnitt führt die grundlegenden formalen Definitionen für die Ablaufsteuerung und die Beschreibung ihrer Algorithmen ein. Die darin definierten Funktionen werden der Ablaufsteuerung von anderen Architekturkomponenten bereitgestellt oder dienen der internen Datenhaltung.

Jedes Crawler-Plugin registriert sich bei der Plugin-Verwaltung mit einer Liste von Komponententypen, die es verarbeiten kann. Für die Ablaufsteuerung bietet die Plugin-Verwaltung die Funktion *kompatiblePlugins* an, die für einen gegebenen Komponententyp eine Liste von Crawler-Plugins zurückgibt, die Komponenten dieses Typs bearbeiten können.

Definition 5.1 (Zum Komponententyp kompatible Plugins)**.** Sei KT die Menge der Komponententypen, sei P die Menge der registrierten Crawler-Plugins, dann ist die Funktion *kompatiblePlugins* definiert als:

$$kompatiblePlugins : KT \rightarrow \wp(P)$$

Ein Crawler-Plugin wird jeweils auf einer Komponente ausgeführt. Falls über diese Komponente hinausgehende Abhängigkeiten auf den ETG bestehen, müssen diese durch das Crawler-Plugin explizit mithilfe der in Abschnitt 4.4 eingeführten Topologie Queries bestimmt werden. Will ein Plugin beispielsweise eine SSH-Verbindung aufbauen, benötigt es die Eigenschaften Hostname, Benutzername und SSH-Zertifikat der Komponente, welche die aktuell vom Crawler-Plugin bearbeitete Komponente betreibt. Die beschriebene Abhängigkeit wird durch einen Topologie Query spezifiziert. Das Ergebnis des Topologie Queries, eine oder mehrere Komponenten, Relationen oder Eigenschaften, wird dem Plugin bereitgestellt. Diese Abhängigkeiten der Crawler-Plugins werden über die Funktion *etgAbhaengigkeiten* der Ablaufsteuerung verfügbar gemacht:

Definition 5.2 (Abhängigkeit von Crawler-Plugins auf den ETG)**.** Sei P die Menge aller Crawler-Plugins, sei *TopologieQueries* die Menge aller möglichen Topologie Queries, dann ist die Funktion *etgAbhaengigkeiten* definiert als:

$$etgAbhaengigkeiten : P \rightarrow \wp(TopologieQueries)$$

Für jedes Element (d.h. eine Komponente oder Relation) und jede Eigenschaft eines Elements führt die Ablaufsteuerung einen Versionszähler, der um eins erhöht wird, wenn ein Crawler-Plugin dieses Element oder dessen Eigenschaft verändert. Falls eine Eigenschaft verändert wird, wird sowohl der Versionszähler der Eigenschaft als auch der des Elements selbst erhöht. Dieser Versionszähler wird verwendet, um festzustellen, ob sich ein Element oder eine Eigenschaft seit der letzten Ausführung eines Crawler-Plugins verändert hat. Zwar werden Plugins nur auf Komponenten ausgeführt, da diese jedoch auch Relationen und Eigenschaften als Abhängigkeiten definieren können (vgl. Definition 5.2), müssen diese auch versioniert sein, um festzustellen, ob sich im für ein Plugin relevanten Teil des ETGs etwas geändert hat. Dieser Versionszähler wird für jede Iteration des Crawler-Laufes gespeichert, weshalb im Folgenden zuerst die Variable *it* eingeführt wird, die jede Iteration eindeutig referenziert.

Definition 5.3 (Crawler-Iteration)**.** Die streng monoton ansteigende Variable $it \in \mathbb{N}_0$ identifiziert jede Iteration eines Crawler-Laufes eindeutig.

Definition 5.4 (Versionszähler)**.** Die Funktion vz_{it} hält die Versionszähler einer Komponente, Relation oder Eigenschaft zum Ende der *it*-ten Iteration des Crawler-Laufes. Sei K die Menge der Komponenten eines ETGs, sei R die Menge der Relationen dieses ETG, sei *Elemente* die Vereinigung aller Komponenten und Relationen, sei N die Menge aller möglichen Namen von Eigenschaften, dann ist die Funktion vz_{it} definiert als:

$$vz_{it} : K \cup R \cup (Elemente \times N) \rightarrow \mathbb{N}_0$$

Die Funktion *la* bestimmt, wann ein Crawler-Plugin das letzte Mal auf einer Komponente ausgeführt wurde.

Definition 5.5 (Letzte Ausführung von Plugin auf Komponente)**.** Sei P die Menge aller Crawler-Plugins, sei K die Menge aller Komponenten eines ETGs, dann ist *la* definiert als:

$$la : (P \times K) \rightarrow \mathbb{N}_0 \cup \{\perp\}$$

Wobei $\perp$ bedeutet, dass dieses Plugin noch nicht auf dieser Komponente ausgeführt wurde.

5.1.5.2 Crawler-Lauf und seine Iterationen

Die Ausführung eines Crawler-Laufes auf einem ETG *etg* wird von Algorithmus 5.1 gesteuert. Dabei werden weitere Iterationen durchgeführt (Zeile 3), solange die vorhergehende Iteration zusätzliche Informationen extrahieren konnte, das heißt, dass sich die Versionszähler geändert haben. Die Schleife endet somit, wenn jeder Versionszähler der aktuellen Iteration *it* den gleichen Wert hat wie in der vorhergehenden Iteration $it-1$ (vgl. Zeile 4). Falls keine neuen Informationen extrahiert werden konnten, terminiert Algorithmus 5.1 und dieser Crawler-Lauf ist beendet.

Algorithmus 5.1 *crawlETG*$(etg \in \mathcal{ETG})$

```
1: it ← 0
2: repeat
3:     crawlerIteration(etg, it)
4: until vz_{it-1} = vz_{it}
```

In jeder Iteration wird durch die Funktion *crawlerIteration* in Algorithmus 5.2 auf jeder Komponente ein oder kein Plugin ausgeführt. Dabei wird in Zeile 1 durch Erhöhen der globalen Variable *it* eine neue Iteration begonnen. In einer Iteration wird für jede Komponente (Zeile 2 von Algorithmus 5.2) durch Aufruf der Funktion *naechstesPlugin* (Zeile 3) das nächste Crawler-Plugin bestimmt, das auf dieser Komponente ausgeführt werden

soll (Zeile 5). Die Funktion *naechstesPlugin* wird detailliert im nächsten Abschnitt (5.1.5.3) behandelt.

Algorithmus 5.2 *crawlerIteration*$(etg \in \mathcal{ETG}, it \in \mathbb{N}_0)$

```
1: it ← it + 1
2: for all k ∈ K_etg do
3:     plugin ← naechstesPlugin(k, it)
4:     if plugin ≠ ⊥ then
5:         plugin auf k ausführen
6:     end if
7: end for
```

Basierend auf Algorithmus 5.1 und Algorithmus 5.2 bietet die Ablaufsteuerung zwei Modi an: (i) einen kompletten Crawler-Lauf durchführen, das heißt, Algorithmus 5.1 *crawlETG* wird verwendet, oder (ii) nur eine Iteration durchführen, das heißt, *crawlerIteration* in Algorithmus 5.2 wird einmal ausgeführt.

5.1.5.3 Auswahl des nächsten Crawler-Plugins

Für jede Iteration *it* bestimmt der Algorithmus 5.3 das nächste Plugin, das auf Komponente *k* ausgeführt werden soll. Die Schleife in Zeile 1 iteriert dazu über alle kompatiblen Crawler-Plugins für den Typ der Komponente *k*. In Zeile 2 werden alle Topologie Queries ausgeführt, welche die Abhängigkeiten des Crawler-Plugins auf den ETG repräsentieren. Daraufhin kann das Plugin entscheiden, ob es mit diesen Ergebnissen arbeiten kann oder nicht (Zeile 3). Diese Entscheidung hängt von dem jeweiligen Plugin ab und kann beispielsweise darauf basieren, ob bestimmte Topologie Queries kein Ergebnis zurücklieferten,[1] diese in der erwarteten Granularität oder dem erwarteten Format vorliegen. Falls dies nicht der Fall ist, wird mit dem nächsten kompatiblen Plugin fortgefahren (Zeile 4). Falls das aktuelle Crawler-Plugin noch nie auf dieser Komponente ausgeführt wurde, wird

[1] Beispielsweise weil die gesuchte Komponente im ETG noch nicht existiert. In einer späteren Iteration, wenn diese Komponente dann existiert, würde das Crawler-Plugin ausgeführt werden, da es dann auf den Daten arbeiten könnte.

Algorithmus 5.3 *naechstesPlugin*($k \in K$, $it \in \mathbb{N}_0$)

```
 1: for all plugin ∈ kompatiblePlugins(typK(k)) do
 2:     d ← Topologie Queries aus etgAbhaengigkeiten(plugin) ausführen
 3:     if plugin kann nicht mit den Daten d arbeiten then
 4:         continue                  // Fortfahren mit nächstem plugin in Zeile 1
 5:     end if
 6:     if la(plugin, k) = ⊥ then     // plugin wurde noch nie auf k ausgeführt?
 7:         return plugin
 8:     end if
 9:     if vz_it(k) ≠ vz_la(plugin,k)(k) then     // k seit letzter Ausführung verändert?
10:         return plugin
11:     end if
12:     for all Komponente/Relation/Eigenschaft e ∈ d do
13:         if vz_it(e) ≠ vz_la(plugin,k)(e) then   // e seit letzter Ausführung verändert?
14:             return plugin
15:         end if
16:     end for
17: end for
18: return ⊥
```

es als Ergebnis zurückgegeben und der Algorithmus endet (Zeilen 6–8). Genauso wird das Plugin zurückgegeben, falls sich die Komponente k seit der letzten Ausführung des Plugins verändert hat, was durch den Vergleich der Versionszähler vz geprüft wird (Zeilen 9–11). Bei Plugins, die in diesem Crawler-Lauf bereits auf der Komponente k ausgeführt wurden und sich die Komponente seitdem nicht geändert hat (d.h. vz ist gleich in Zeile 9), wird nun überprüft, ob sich die Ergebnisse der Topologie Queries seit der letzten Ausführung des Plugins verändert haben (Zeilen 12–16). Dazu wird in den Zeilen 13–15 für jedes Ergebnis eines Topologie Queries (d.h. eine Komponente, Relation oder Eigenschaft, vgl. Abschnitt 4.4) überprüft, ob sich sein Versionszähler geändert hat. Es werden also nur die Versionszähler und nicht die Ergebnisse der Topologie Queries gespeichert. Falls dies für mindestens ein Ergebnis der Topologie Queries zutrifft, wird dieses Plugin zurückgegeben. Damit wird sichergestellt, dass ein Plugin ausgeführt wird, wenn sich seine Datenbasis verändert hat, also entweder die Komponente k selbst oder die Ergebnisse der Topologie Queries. Falls es für den Typ

von Komponente *k* kein kompatibles Plugin gibt oder falls alle kompatiblen Crawler-Plugins schon auf dieser Version der Daten ausgeführt wurden, wird $\perp$ zurückgegeben (Zeile 18).

5.1.5.4 Selektives Crawling des ETGs

In manchen Anwendungsfällen, beispielsweise der Variante 1 („Selektive Erstellung der Enterprise Topologie“) der AROMA-Methode, ist es nötig, nur einen ausgewählten Ausschnitt des ETGs zu erstellen. Dabei müssen verschiedene, vom Anwendungsfall abhängige, Kriterien für die Einschränkung des Ausschnitts definierbar sein. Außerdem sollen keine zusätzlichen Anforderungen an Crawler-Plugins gestellt werden, das heißt, auf das selektive Crawling soll bei der Entwicklung von Crawler-Plugins keine Rücksicht genommen werden müssen. Dieser Abschnitt beschreibt eine Erweiterung der Ablaufsteuerung, die das selektive Crawling von ETGs unter den zuvor genannten Voraussetzungen erlaubt.

Das Grundkonzept der gewählten Lösung ist, den ETG nach jeder Iteration *zurechtzuschneiden* und so sicherzustellen, dass nur der gewünschte Ausschnitt enthalten ist. Dies wird von der Ablaufsteuerung durch wiederholtes Anwenden der Funktion *selektionSicherstellen* durchgeführt, welche die Elemente des ETGs außerhalb des gewünschten Ausschnittes löscht. Je nach Anwendungsfall oder gewünschtem Kriterium kann eine andere Funktion *selektionSicherstellen* verwendet werden. Diese Lösung verlangt keinen Eingriff in die Crawler-Plugins, weil diese weiterhin nur auf den von der Ablaufsteuerung übergebenen Komponenten arbeiten. Auch wird sichergestellt, dass nicht mehr als die Arbeit einer Iteration an den Komponenten außerhalb des gewünschten Ausschnittes verworfen wird. Der Grund dafür ist, dass ein Crawler-Plugin nur auf einer Komponente ausgeführt wird, wenn sich diese, bzw. die anderen Abhängigkeiten des Crawler-Plugins auf den ETG, verändert haben. Wenn ein Plugin eine Komponente außerhalb des gewünschten Ausschnitts erstellt, wird diese verworfen und nicht in der nachfolgenden Iteration wieder erstellt. Darüber hinaus ist gewährleistet, dass nie ein Crawler-Plugin auf einer Komponente außerhalb des gewünsch-

ten Ausschnitts ausgeführt wird. Algorithmus 5.4 zeigt die Abwandlung der Funktion *crawlETG* in Algorithmus 5.1 für das selektive Crawling des ETGs.

Algorithmus 5.4 *crawlETGSelektiv*($etg \in \mathcal{ETG}$)

```
1: it ← 0
2: repeat
3:     crawlerIteration(etg, it)
4:     selektionSicherstellen(etg)
5: until vz_{it-1} = vz_{it}
```

Zur Realisierung der Variante 1 („Selektive Erstellung der Enterprise Topologie“) der AROMA-Methode soll ein ETG gecrawlt werden, der ausschließlich die Komponenten enthält, die für den Betrieb der vom Nutzer bestimmten Komponenten mit Anwendungsfunktionalität nötig sind. Dies ist die gleiche Problemstellung wie bei der Identifikation der zu migrierenden Komponenten, die in Abschnitt 4.5 behandelt wurde. Deshalb kann hier die Funktion *selektionSicherstellen* durch die Funktion *deepDive* aus Algorithmus 4.3 realisiert werden. Der Parameter $K_{eingabe}$ von *deepDive* ist dabei die Menge der Komponenten mit Anwendungsfunktionalität, für welche der Ausschnitt des ETGs erstellt werden soll. Diese Komponenten werden damit sowohl als Einstiegskomponenten für den Crawler als auch für die Sicherstellung des richtigen Ausschnitts verwendet.

Vorteil des selektiven Crawlings von ETGs ist der entsprechend deutlich reduzierte Aufwand, falls nur ein bestimmter Ausschnitt des ETGs relevant ist. Damit kann die Migration einzelner Anwendungen schneller durchgeführt werden, ohne auf das Crawling des gesamten ETGs warten zu müssen. Auch in anderen Anwendungsfällen, zum Beispiel bei der automatisierten Verwaltung existierender Anwendungen, wird das selektive Crawling verwendet [BBK+14b, BBKL14c]. Die gewählte Lösung benötigt keine Berücksichtigung des selektiven Crawlings in den Crawler-Plugins und erlaubt es, eine vom Anwendungsfall abhängige Implementierung der Funktion *selektionSicherstellen* zu verwenden. Das in diesem Abschnitt beschriebene Vorgehen hat jedoch den Nachteil, dass in manchen Fällen eingehende Relationen zu Komponenten im selektierten Ausschnitt nicht gefunden wer-

den. Dies ist der Fall für alle Relationen, die nur *in Richtung der Relation* durch Crawler-Plugins erkannt werden können. Dann wird, wenn eine Komponente nicht im selektierten Ausschnitt ist, diese Relation nicht im ETG repräsentiert. Diese Einschränkung muss bei der Anwendung des selektiven Crawlings beachtet werden.

5.1.5.5 Bestehenden ETG aktualisieren

Da die Crawler-Methode die IT *von außen* betrachtet,[1] ohne dass die Komponenten davon wissen, wird die Ablaufsteuerung auch nicht über Änderungen der IT informiert. Um den ETG aktuell zu halten, wie in Anforderung CA-4 („Aktualisierung von ETGs“) gefordert, muss regelmäßig oder manuell bei Bedarf (je nach Anforderung des Anwendungsfalls) eine Aktualisierung durchgeführt werden. Dafür wird die Funktion *la*, wie in Algorithmus 5.5 beschrieben, zurückgesetzt, wodurch alle Crawler-Plugins nochmals ausgeführt werden, der eigentliche ETG aber erhalten bleibt. Zur Aktualisierung des ETGs werden dieselben Crawler-Plugins und dieselbe Ablaufsteuerung verwendet wie für das Erstellen des ETGs. Jedoch sinkt der Aufwand durch die Wiederverwendung der meisten Daten deutlich, da nicht alle Daten neu extrahiert werden. Zusätzlich können Crawler-Plugins Maßnahmen ergreifen, um den Aufwand der Aktualisierung weiter zu reduzieren, beispielsweise indem sie nur prüfen, ob sich an der entsprechenden Komponente etwas geändert hat. Ein Crawler-Plugin für den Tomcat Webserver könnte beispielsweise die Liste der installierten Anwendungen laden und diese mit dem ETG abgleichen. Basierend darauf würden dann die Anwendungskomponente und Relationen entsprechend hinzugefügt oder entfernt.

5.1.5.6 Rückfragen an den Crawler-Administrator

Für den Fall, dass ein Crawler-Plugin Daten benötigt, die bisher nicht im ETG enthalten sind, zum Beispiel besondere Zugangsdaten, kann es eine Rückfrage an den Crawler-Administrator stellen. Zur Bearbeitung der Rückfrage

[1] Dies wird impliziert durch Anforderung CA-5 („Einfluss auf Produktionssysteme minimieren“), die fordert, dass Produktionssysteme für das Crawling nicht verändert werden.

Algorithmus 5.5 *etgAktualisieren*($etg \in \mathcal{ETG}$)

```
1: for all plugin ∈ P do
2:     for all k ∈ K_etg do
3:         la(plugin, k) ← ⊥
4:     end for
5: end for
6: crawlETG(etg)                          // bzw. crawlETGSelektiv(etg)
```

wird der Crawler-Lauf nach Ende der aktuellen Iteration pausiert (nachdem alle Crawler-Plugins ausgeführt wurden), damit der Crawler-Administrator die manuellen Eingaben gesammelt tätigen kann. Anschließend wird der Crawler-Lauf fortgesetzt, mit der Besonderheit, dass die Crawler-Plugins, welche Rückfragen gestellt haben, auf der entsprechenden Komponente nochmals ausgeführt werden, so dass diese die erfragten Informationen verarbeiten können. Crawler-Plugins sollten die Funktion zur Rückfrage möglichst sparsam verwenden, weil dies den Aufwand, die Laufzeit und damit auch die Kosten eines Crawler-Laufes erhöht.

5.1.6 Deduplizierung und Konsolidierung

Aufgrund der verschiedenen Informationsquellen und da Crawler-Plugins sich nicht gegenseitig kennen, ist es nötig, Duplikate zu verhindern und Daten zu konsolidieren, um die Qualität des ETGs sicherzustellen (vgl. Anforderung CA-1). Duplikate im ETG können bei einem Crawler-Lauf auftreten, wenn die gleiche Komponente auf unterschiedlichen Wegen oder von verschiedenen Plugins gefunden wird. Zum Beispiel kann eine virtuelle Maschine unter ihrer IP-Adresse (z.B. `192.168.209.195`) und ihrem Domainnamen (z.B. `www.example.org`) gefunden werden oder ein Dienst wird von zwei Komponenten genutzt und deshalb fälschlicherweise als zwei separate Komponenten repräsentiert. Auf der anderen Seite kann die Zusammenführung von zwei ETGs nötig sein, die beispielsweise parallel in unterschiedlichen Teilen bzw. isolierten Netzwerken der IT erstellt wurden. Für die Erkennung und Beseitigung von Duplikaten gibt es zwei grundsätz-

liche Herangehensweisen: (i) die inkrementelle Deduplizierung, welche Duplikate verhindert, indem alle Operationen auf dem ETG geprüft werden, und (ii) die Deduplizierung des kompletten ETGs, das heißt, Duplikate werden zugelassen und später beseitigt.

Im nachfolgenden Abschnitt 5.1.6.1 werden bestehende Ansätze zur kompletten Deduplizierung betrachtet, die auf den ETG adaptiert werden können. Diese Art der Deduplizierung wird aber darüber hinaus nicht weiter verfolgt. Abschnitt 5.1.6.2 stellt eine Lösung für die inkrementelle Deduplizierung vor, die auf Ansätzen der in Abschnitt 5.1.6.1 vorgestellten Arbeiten aufbaut. Die inkrementelle Deduplizierung ist im Kontext des Crawlers interessant, da schon während eines Crawler-Laufes Duplikate vermieden werden und somit verhindert wird, dass auf diesen weitergearbeitet wird und beispielsweise identische Informationen mehrfach extrahiert werden. Außerdem soll es möglich sein, den Crawler-Lauf jederzeit zu pausieren und trotzdem ein konsistentes Abbild der IT vorliegen zu haben.

5.1.6.1 Komplette Deduplizierung

Die Deduplizierung eines kompletten ETGs bzw. die Zusammenführung zweier ETGs kann verallgemeinert und auf das Zusammenführen von Graphen abgebildet werden. Für das allgemeine Problem gibt es bereits eine große Anzahl an Arbeiten, beispielsweise im Forschungsbereich des Data-Minings und -Cleanings, der Knowledge Discovery oder des Information Retrieval. Newcombe et al. [NKAJ59] beschreiben bereits 1959, wie Einträge aus Personenstandsbüchern (Geburten, Heiraten, Sterbefälle) zusammengeführt werden können. Dazu werden basierend auf dem „Soundex Code“[1] des Nachnamens, des Geburtsnamens und des Geburtsdatums Metakanten zwischen Einträgen potentiell gleicher Personen erstellt. Es werden also verschiedene Eigenschaften der Person zusammengefügt, um eine eindeutige ID zu erstellen. „Open Services for Lifecycle Collaboration“ (OSLC) nutzt ein ähnliches Verfahren, um Entitäten beim Suchen und Navigieren zu grup-

[1] Soundex beschreibt Berechnungsregeln, um amerikanischen Namen einen Code zuzuweisen, der bei gleich oder ähnlich ausgesprochenen Namen identisch ist [The07].

pieren, zum Beispiel die Monitoring-Informationen aus einem System mit einem anderen System zu verlinken. Dazu beschreibt eine Unterspezifikation [Dan13], welche Eigenschaften bei welchem Typ von Entität vorhanden sein müssen, um diese mit anderen Entitäten gleichen Typs zu korrelieren. Iyengar et al. [IJCK06] haben ein auf Adaptern basiertes System erstellt, das Personen, Firmen, Mails und wissenschaftliche Veröffentlichungen mithilfe von Adaptern zusammenführt. Der „Person-zu-Person"-Adapter bestimmt beispielsweise, ob zwei Einträge die gleiche Person beschreiben, und erstellt entsprechende Relationen, die für die Navigation und das Suchen genutzt werden. Andere Ansätze, wie von Parsons und Wand [PW03] oder Domingos [Dom04], gehen über das starre Vergleichen der Eigenschaften hinaus und betrachten den Grad der Ähnlichkeit aller Eigenschaften zweier Entitäten als gewichtete Relation. Dong et al. [DHM05] nutzen einen Graphen, um den Grad der Ähnlichkeit von Entitäten darin zu propagieren, ähnlich wie dies Googles „PageRank" [PBMW99] macht. Eine detailliertere Übersicht offener Forschungsfragen in diesem Gebiet hat Winkler [Win99] erstellt. Ein Konzept zur semantisch korrekten Zusammenführung von TOSCA-Awendungstopologien wurde in [BBK+13] veröffentlicht.

Es bleibt festzuhalten, dass es eine Vielzahl von existierenden Arbeiten gibt, welche die Ähnlichkeit von Entitäten bestimmen – meistens basierend auf einer Teilmenge der Eigenschaften. Dabei werden die Eigenschaften teilweise nicht direkt verglichen, sondern nur ein abgeleiteter Wert, der besser vergleichbar ist, zum Beispiel die Aussprache bei Newcombe et al. [NKAJ59]. Die in diesem Abschnitt beschriebenen Arbeiten deduplizieren den aktuellen Datenbestand, im Gegensatz zur im nächsten Abschnitt vorgestellten inkrementellen Deduplizierung, die alle Änderungsoperationen prüft.

5.1.6.2 Inkrementelle Deduplizierung

Das grundlegende Konzept der inkrementellen Deduplizierung ist, dass der ETG immer duplikatfrei bleibt. Dazu muss einerseits erkannt werden, ob eine Operation auf dem ETG zu einem Duplikat führen würde, und andererseits ein Mechanismus entwickelt werden, der dies verhindert, ohne dass Daten

verloren gehen. Diese Prüfung könnte in der Implementierung des jeweiligen Crawler-Plugins durchgeführt werden, beispielsweise indem Plugins immer den gesamten ETG betrachten. Dies bringt jedoch eine Reihe von Nachteilen mit sich: (i) Die Plugins würden mehrfach die gleiche Funktionalität implementieren, (ii) die Komplexität von Crawler-Plugins und somit der Aufwand für deren Erstellung würde steigen und (iii) dies wäre ein Verstoß gegen das Architekturprinzip, dass Crawler-Plugins ausschließlich typspezifische Logik zum Extrahieren von Informationen enthalten und nicht zur Verwaltung des ganzen ETGs. (iv) Darüber hinaus kann der Crawler den Plugins nicht vertrauen, diese Funktionalität (fehlerfrei) umzusetzen, da diese von beliebigen, auch Dritten, entwickelt werden können (vgl. Anforderung CA-2: „Erweiterbarkeit“). Aus diesen Gründen wurde eine separate Architekturkomponente „Deduplizierung“ eingefügt, die zwischen den Plugins und den Operationen auf dem ETG steht (vgl. Abschnitt 5.1.3).

Die Architekturkomponente zur Deduplizierung kann für jede Komponente eine *typabhängige Identität* (im Weiteren abgekürzt mit „ID“), basierend auf den Eigenschaften der Komponente, berechnen. Dies ist möglich, da der ETG ein Typsystem hat, das die Eigenschaften der Komponenten und deren Semantik beschreibt. Gebildet wird eine ID durch die Konkatenation eines oder mehrerer Werte der Eigenschaften der Komponente, welche durch ein Trennzeichen separiert werden. Die Namen der zur Bildung der ID genutzten Eigenschaften und deren Reihenfolge werden im Typsystem definiert. Für eine virtuelle Maschine kann das die IP-Adresse sein, für einen Webserver Hostname und Port und für einen Webservice die komplette URL. Dies ist vergleichbar mit einem zusammengesetzten Primärschlüssel bei relationalen Datenbanken oder den zuvor vorgestellten Ansätzen zur kompletten Deduplizierung [NKAJ59, Dan13]. Es ist jedoch möglich, dass eine Eigenschaft der ID verschiedene Formate des Datums oder verschiedene Arten von Werten zulässt. Beim Host des Webservers ist beispielsweise sowohl die IP-Adresse als auch der Hostname ein zulässiger Wert der Eigenschaft, womit zwei Komponenten vom Typ Webserver den gleichen Webserver beschreiben könnten, ohne die gleiche ID zu haben. Dies wird adressiert, indem die generische Methode zur Erstellung der ID in diesen Fällen durch eine typspe-

zifische Logik überschrieben werden kann, wie von Iyengar et al. [IJCK06] vorgeschlagen. Für den Webserver könnte die IP-Adresse als ID definiert und der Hostname entsprechend aufgelöst werden.

Falls bereits eine Komponente k_a mit der gleichen ID existiert, werden die Daten der neuen Komponente k_b in k_a übernommen, damit möglicherweise neue Informationen nicht verloren gehen. Generisch können alle Eigenschaften von k_b nach k_a kopiert werden und bei gleichem Namen der Wert der Eigenschaft von k_a überschrieben werden. Aufgrund des offenen Typsystems kann nicht generisch geprüft werden, welcher der beiden Werte genauer ist, weil die Semantik der Werte nicht bekannt ist. Auch hier ist es möglich, diese generische Methode durch eine typspezifische Logik zu ersetzen, die beispielsweise auch Berechnungen anstellen kann, anstatt sich einfach nur für einen der Werte zu entscheiden.

Die typspezifische Logik wird über separate Plugins der Deduplizierungskomponente bereitgestellt, wenn die zuvor beschriebenen generischen Ansätze für einen Komponententyp nicht ausreichen. Damit wird die Logik für die Deduplizierung komplett von den Crawler-Plugins getrennt und folglich sichergestellt, dass diese Logik nur einmal implementiert werden muss. Der hier beschriebene Ansatz zur Deduplizierung wurde im Prototyp der vorliegenden Arbeit angewandt und implementiert.

5.2 Arbeitsweise von Crawler-Plugins

Dieser Abschnitt stellt detailliert die Funktionsweise einer Auswahl von Crawler-Plugins dar. Im Gegensatz zu Abschnitt 5.1, der die Steuerung eines Crawler-Laufes und die Zusammenarbeit von Crawler-Plugins beschreibt, liegt der Schwerpunkt dieses Abschnitts auf der Realisierung und Implementierung von konkreten Crawler-Plugins. Ziel ist es, zu veranschaulichen, wie Crawler-Plugins Informationen extrahieren und auf welchen Wegen dies geschehen kann. Ein Überblick aller 28 bisher entwickelten Crawler-Plugins wird in Abschnitt 7.3.3 gegeben, der den Prototyp des Crawlers und dessen Architektur beschreibt.

5.2.1 BPEL und WSDL

Für einen Crawler-Lauf auf einem BPEL-Geschäftsprozess ist eine Komponente vom Typ *BPEL* gegeben, an die der Geschäftsprozess als Datei angehängt ist. Zu Beginn des Crawler-Laufes wird das entsprechende Crawler-Plugin aufgerufen, das den BPEL-Geschäftsprozess lädt und analysiert. Dabei wird für jeden im Geschäftsprozess verwendeten PortType eine Komponente vom Typ *WSDL_PortType* angelegt und durch eine Relation vom Typ *calls* (Kindtyp von *connectsTo*; vgl. Abschnitt 4.2.5) mit der BPEL-Komponente verbunden. Dazu verarbeitet das Plugin jeden in BPEL definierten PartnerLink und bestimmt mit dessen PartnerLinkType sowie den Eigenschaften `MyRole` und/oder `PartnerRole` den vom Prozess aufgerufenen PortType und die WSDL-Datei, die diesen PortType definiert. Die Informationen des PartnerLink und PartnerLinkType werden in der Relation zwischen BPEL- und PortType-Komponente als Eigenschaften hinterlegt.

In der nächsten Iteration des Crawler-Laufes wird auf den Komponenten vom Typ *WSDL_PortType* das dem Typ entsprechende Crawler-Plugin ausgeführt. Dieses bestimmt für den PortType alle Implementierungen, die in den dem Crawler-Plugin bekannten WSDL-Dateien definiert sind, beispielsweise in von der BPEL-Datei importierten WSDL-Dateien. Dazu werden zuerst alle Bindings, die eine Umsetzung des PortType definieren, bestimmt und anschließend alle Services nach Ports durchsucht, die diese Bindings realisieren. Für jeden Port wird eine Komponente vom Typ *Application* erstellt, da im ETG im Rahmen der vorliegenden Arbeit nicht zwischen Diensten und Anwendungen unterschieden wird. In diese Komponenten werden abhängig von der Art des Bindings verschiedene Informationen als Eigenschaften gespeichert. Beim „SOAP-HTTP“-Binding wird unter anderem die Eigenschaft `URL` gesetzt. Diese Komponenten werden durch eine Relation vom Typ *implementedBy* (Kindtyp von *hostedOn*; vgl. Abschnitt 4.2.5) mit der Komponente, die den PortType repräsentiert, verbunden. Die Relation enthält Informationen zum WSDL-Binding und WSDL-Service als Eigenschaften. Basierend auf dieser Komponente vom Typ *Application* und dem Binding „SOAP-HTTP“ beschreibt Abschnitt 5.2.3 das weitere Vorgehen, wenn dieser Dienst durch

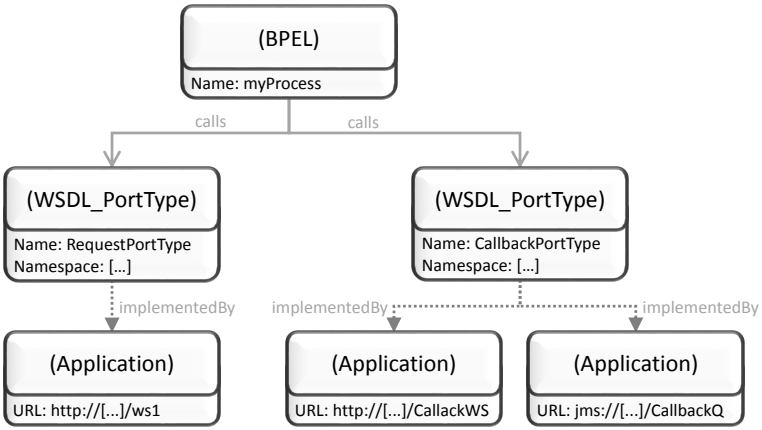

Abbildung 5.3: ETG-Repräsentation eines BPEL-Geschäftsprozesses

einen ESB bereitgestellt wird, und Abschnitt 5.2.2, falls der Dienst direkt unter dieser URL erreichbar ist.

Abbildung 5.3 zeigt einen exemplarischen ETG, der durch die in diesem Abschnitt diskutierten Crawler-Plugins erstellt wurde. Dabei werden die oben erwähnten Eigenschaften der Relationen nicht dargestellt.

Im Prototyp der vorliegenden Arbeit wurden, wie in diesem Abschnitt beschrieben, der BPEL-Geschäftsprozess, PortType und der implementierende Dienst als Komponente und das WSDL-Binding, -Port und -Service als Relation im ETG repräsentiert. Da jeder Port, bzw. der dahinter liegende Dienst, alle Operationen eines PortTypes implementiert, wurde als Granularität zur Darstellung der technischen Implementierung des Geschäftsprozesses im ETG der PortType gewählt. Eine oder mehrere ausgehende Relationen der Komponente, die den PortType repräsentiert, zeigen die verschiedenen in der WSDL definierten Implementierungen (d.h. Ports). Auf der rechten Seite von Abbildung 5.3 wird ein PortType mit zwei verschiedenen Bindings dargestellt. Ein ETG kann nach Definition 4.1 mehrere Relationen für verschiedene Bindings zwischen einem PortType und Dienst enthalten. Bei der Migration kann dann beispielsweise entschieden werden, dass nur eine Implementierung des PortTypes migriert werden soll.

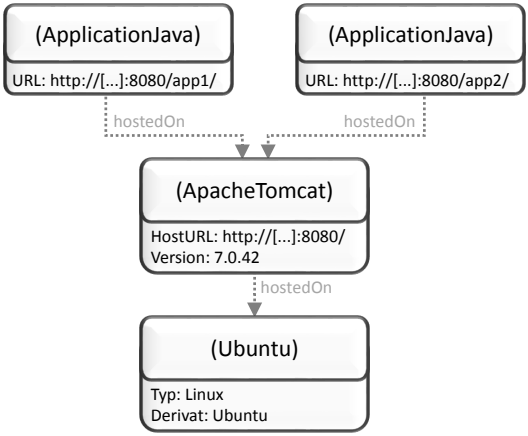

Abbildung 5.4: ETG-Repräsentation Webanwendung und Webservers

5.2.2 Anwendung und Webserver

Zur Beschreibung der Crawler-Plugins für Anwendungen und Webserver ist eine Komponente vom Typ *Application* gegeben. Da es sich um eine Webanwendung handelt, ist die Eigenschaft `URL` gesetzt. Aufgrund des Typs der Komponente wird zu Beginn des Crawler-Laufes zuerst das generische Crawler-Plugin für Anwendungen ausgeführt, welches die als Eigenschaft definierte URL abruft (HTTP GET) und die Antwort untersucht. Durch die Analyse des standardisieren HTTP-Headers `Server` [FGM+97] oder des nicht-standardisierten, aber weitverbreiteten HTTP-Headers `X-Powered-By` ist es möglich zu unterscheiden, ob es sich zum Beispiel um eine Java-, .NET- oder PHP-Anwendung handelt. In diesem Beispiel wird deshalb der Typ der Komponente auf `ApplicationJava` geändert, wie in Abbildung 5.4 gezeigt. Das Abrufen der URL geschieht, ohne dabei zu wissen, um was für eine Art von Anwendung es sich handelt, da beispielsweise auch eine Fehlermeldung aufgrund einer nicht vorhandenen Authentifizierung diese HTTP-Header enthält. In der nächsten Iteration wird das allgemeine Crawler-Plugin für Komponenten vom Typ *ApplicationJava* aufgerufen, das ausschließlich eine Komponente vom Typ *Webserver* anlegt und die Eigenschaft `HostURL` für die

Weiterverarbeitung in nachfolgenden Plugins setzt. Die Java-Anwendung wird dann mit einer ausgehenden Relation vom Typ *hostedOn* mit dem Webserver verbunden.

Das generische Crawler-Plugin für Komponenten vom Typ *Webserver* lädt, ähnlich wie zuvor bei der Anwendung, diejenige Ressource, die unter der URL in der Eigenschaft `HostURL` erreichbar ist. Falls der HTTP-Header `Server` vorhanden und dem Plugin dessen Wert bekannt ist, wird der Typ der Komponente entsprechend geändert, in diesem Beispiel auf `ApacheTomcat`. Aus Sicherheitsgründen ist dieser Header jedoch teilweise nicht vorhanden, zum Beispiel weil der Betreiber den Typ des Webservers verschleiern will, um die Ausnutzung von Sicherheitslücken zu erschweren. In diesem Fall wird das externe Werkzeug „Httprint“ [Net05] aufgerufen, das einen Fingerabdruck des Webservers erstellt (vgl. Abschnitt 2.6.5). Basierend auf einer Datenbank von bekannten Fingerabdrücken wird der Typ des Webservers bestimmt und die ETG-Komponente entsprechend angepasst. Dieses Werkzeug wird integriert, indem es auf der Kommandozeile des Betriebssystems, auf dem der Crawler läuft, ausgeführt wird und die Ausgaben analysiert werden.

Um weitergehende Informationen über die Tomcat-Komponente zu ermitteln, stehen drei Informationsquellen mit unterschiedlichen Zugriffswegen zur Verfügung [Apa14c]: (i) die Tomcat Manager REST Schnittstelle, (ii) die Tomcat JMX Schnittstelle und (iii) die Tomcat-Konfigurationsdatei (`server.xml`). Da jede dieser Informationsquellen Informationen in einem anderen Format bereitstellt und die verlange Authentifizierungen an den Informationsquellen unterschiedlich ist, wurde für jede Datenquelle ein separates Crawler-Plugin entwickelt. Neben der präzisen Tomcat-Version, `Apache Tomcat/7.0.42` in diesem Beispiel, sowie Informationen über das Betriebssystem und die Java-Laufzeitumgebung bieten die Informationsquellen (i) und (ii) eine Liste aller installierten Anwendungen. Die ermittelten Informationen werden als Eigenschaften der Komponente hinzugefügt bzw. es wird im Falle der installierten Anwendungen für jede Anwendung eine entsprechende Komponente vom Typ *ApplicationJava* angelegt. Falls so Informationen über das darunterliegende Betriebssystem gewonnen werden, normalerweise Hersteller, Typ, Version, Prozessorarchitektur und Ähnli-

ches, wird eine entsprechende Komponente angelegt und mit einer von der Tomcat-Komponente ausgehenden Relation vom Typ *hostedOn* verbunden.

Die Tomcat-Konfigurationsdatei (iii) kann erst ausgewertet werden, wenn das darunterliegende Betriebssystem gecrawlt wurde (siehe nachfolgenden Abschnitt 5.2.4) und ein Zugriff auf das Dateisystem möglich ist, zum Beispiel per SSH, oder wenn der Crawler-Administrator diese Datei zur Verfügung stellt. Das Crawler-Plugin fügt diese Konfigurationsdatei der Tomcat-Komponente im ETG hinzu, so dass diese auch von weiteren Plugins analysiert werden oder für eine Bereitstellung mit gleicher Konfiguration in einer anderen Umgebung genutzt werden kann. In gleicher Weise abhängig vom Betriebssystem ist das Plugin „ApplicationJavaSSH“, das für jede Komponente, die eine Java-Anwendung repräsentiert, das entsprechende Java Web Archive (WAR) extrahiert und als Datei an die entsprechende Komponente anhängt.

5.2.3 Enterprise Service Bus

Bei der Umsetzung einer Service-orientierten Architektur (vgl. Abschnitt 2.1) spielen Enterprise Service Busse (ESB) eine zentrale Rolle [Cha04]. Am ESB registrieren sich alle verfügbaren Dienste und alle Aufrufe dieser Dienste durchlaufen den ESB, wobei der ESB den Dienst auswählt und den Aufruf möglicherweise auch transformiert. Durch die Verwendung von ESBs wird jedoch eine Indirektion zwischen einem Dienst und dessen Konsumenten eingeführt. Diese Indirektion ist im Kontext der Serviceorientierung gewünscht, führt aber zu Problemen bei der Repräsentation im ETG, weil nicht mehr ersichtlich ist, welche Komponenten (d.h. Konsumenten) welche anderen Komponenten (d.h. Dienste) nutzen. Um dies zu adressieren, wurde eine abstrakte Repräsentation von ESB-Routen im ETG entwickelt, in Anlehnung an die „Enterprise Integration Patterns“ (EIP) von Hohpe und Woolf [HW03]. Darauf basierend wurde eine Reihe von Crawler-Plugins entwickelt, welche die für die Repräsentation des ESB und dessen Routen im ETG nötigen Informationen extrahieren. Ziel ist es dabei, sowohl statisches Routing durch Analyse der Konfiguration als auch, wo möglich, die

letztendliche dynamische Dienstauswahl durch die Nutzung von Statistiken abzudecken. Im resultierenden ETG sollten alle potentiell von einem Konsumenten verwendeten Dienste, sprich ETG-Komponenten, sowie die Wahrscheinlichkeit, dass ein bestimmter Dienst aufgerufen wird, als Gewichtung der Relationen enthalten sein. Dieser Abschnitt geht zuerst auf die Repräsentation von ESBs im ETG und anschließend auf die Crawler-Plugins ein, welche die Informationen aus den beiden Open-Source-ESBs „Apache Synapse“ [Apa11] und „Apache Camel“[1] [Apa14a] extrahieren.

Basierend auf den „Enterprise Integration Patterns“ (EIP) [HW03] wurde eine ESB-unabhängige Repräsentation von ESBs und deren Routing entwickelt, welche die Typen von Komponenten und Relationen im ETG definiert. Dabei wird nicht die gesamte Bandbreite an EIPs im ETG abgebildet, sondern eine dem Anwendungsfall entsprechende Auswahl und Abstraktion, die repräsentiert, welche Dienste von einem Konsumenten verwendet werden. Zum Beispiel die Dienste, die ein Geschäftsprozess oder eine Anwendung potentiell benutzt, die eine Nachricht an einen ESB schicken. Dazu werden die vier Komponententypen (i) *ESBRoute*, (ii) *Application*, (iii) *Transformator* und (iv) *MessageRouter* verwendet bzw. definiert.

Eine Komponente vom Typ *ESBRoute* repräsentiert den Dienst, den der ESB für Konsumenten anbietet. Nachrichten die an diesen Dienst gesendet werden verarbeitet der ESB, entsprechen der Konfiguration der Route. Das Ende der ESB-Route ist eine Komponente vom Typ *Application*, an die der ESB ausgehende Nachrichten weiterleitet.

Der Typ *Application* ist, wie schon zuvor eingeführt, der Obertyp zur Repräsentation aller Komponenten mit Anwendungsfunktionalität. Falls mehr Informationen über den Typ vorhanden sind, beispielsweise, dass es sich um einen Java Webservice handelt, wird ein entsprechender Kindtyp verwendet. Diese Komponente kann auch rekursiv wieder ein Einstiegspunkt in eine andere ESB-Route sein, dann wäre der Typ entsprechend eine *ESBRoute*. Die

[1] Apache Camel ist strenggenommen kein ESB, bildet aber die Basis für den ESB „Apache ServiceMix“ (`http://servicemix.apache.org/`) und ermöglicht die Entwicklung von Java-Anwendungen, die ESB-ähnliche Routing- und Transformationsmöglichkeiten enthalten (`http://camel.apache.org/java-dsl.html`).

erstellten Komponenten vom Typ *Application* werden von den im vorherigen Abschnitt 5.2.2 beschriebenen Crawler-Plugins weiter bearbeitet. Diese beiden Komponententypen repräsentieren damit die Gruppe der *Messaging Endpoint* EIPs [HW03], wobei Hohpe und Woolf nicht zwischen dem Einstiegspunkt, mit dem der Konsument kommuniziert, und dem Endpunkt, an den der ESB die Nachricht letztendlich sendet, unterscheiden.

Die Gruppe *Message Transformation* der EIPs [HW03] (z.B. *Canonical Data Model* und *Content Filter*) wird durch eine Komponente vom Typ *Transformator* repräsentiert, die als Eigenschaften die entsprechenden Details der Transformation enthält. Die Gruppe *Message Routing* der EIPs [HW03] (z.B. *Content-Based Router* und *Message Filter*) wird durch eine Komponente vom Typ *MessageRouter* repräsentiert: Ein MessageRouter kann eine eingehende Nachricht zwischen verschiedenen nachfolgenden Komponenten verteilen und aufteilen.

Die Relationen zwischen den ESB-Komponententypen repräsentieren den Nachrichtenfluss sowie Bedingungen für das Routing als Eigenschaften. Sie repräsentieren also die *Messaging Channels* Gruppe der EIPs [HW03]. Statistiken, beispielsweise die Anzahl der Aufrufe bestimmter Komponenten der Route, werden als zusätzliche Eigenschaften der Komponente gespeichert.

Abbildung 5.5 zeigt eine beispielhafte ESB-Route: Zuerst teilt eine Komponente vom Typ *MessageRouter* die Nachrichten in solche aus dem privaten und öffentlichen Netz. Dann werden die Nachrichten transformiert und an den verarbeitenden Dienst weitergeleitet. Nachrichten aus dem öffentlichen Netz werden dabei gleichmäßig über zwei Instanzen des gleichen Dienstes verteilt. Dieses Beispiel zeigt, dass trotz der Abstraktion der Konfiguration des ESBs im ETG ersichtlich ist, welche Dienste von welchem Konsumenten genutzt werden.

Zu Beginn des Crawlings muss festgestellt werden, ob es sich bei einer Komponente vom Typ *Application* um einen Endpunkt handelt, der von einem ESB bereitgestellt wird, oder nicht. Dazu wird das entsprechende Crawler-Plugin für diesen Komponententyp erweitert. Wie in Abschnitt 5.2.2 beschrieben werden, falls die Eigenschaft `URL` definiert ist, die HTTP-Header analysiert, um weitergehende Informationen über die Anwendung zu erhal-

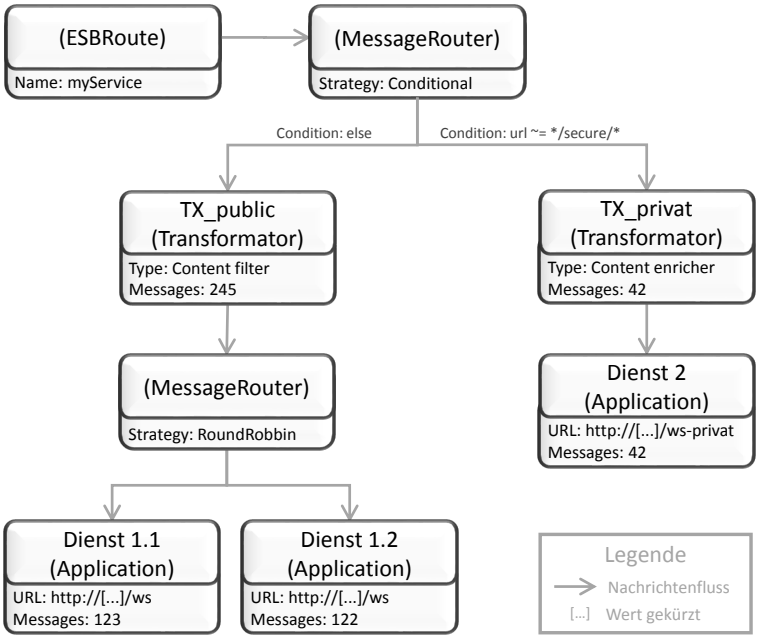

Abbildung 5.5: ETG-Repräsentation einer Beispiel-ESB-Route

ten. Dies wird auch angewandt, um HTTP-Endpunkte, die von den ESBs Apache Camel und Apache Synapse bereitgestellt werden, zu erkennen und der Komponente den entsprechenden Kindtyp vom Typ *ESBRoute* zuzuweisen. Bisher wurde das Crawler-Plugin nur für HTTP-Endpunkte implementiert, ein ähnliches Vorgehen ist jedoch auch bei anderen Transport-Protokollen möglich. Im Folgenden werden die ESB-spezifischen Crawler-Plugins sowie deren Vorgehen beim Extrahieren der Daten vorgestellt.

Apache Camel bietet eine JMX-Schnittstelle an, um alle Routen der laufenden Instanz in einem einheitlichen Format abzurufen, obwohl Apache Camel die Routendefinition auf verschiedenen Wegen zulässt, zum Beispiel in einer Java DSL oder XML [Apa14a]. Darüber hinaus stellt die JMX-Schnittstelle auch statistische Informationen bereit. Auf die Details der Transformation der Java-Objekte, welche über die JMX-Schnittstelle bereitgestellt werden, in die ESB-unabhängige Repräsentation im ETG wird hier verzichtet.

Apache Synapse nutzt XML-Konfigurationsdateien, die lokal gespeichert, aber nicht nach außen zur Verfügung gestellt werden. Diese Konfigurationsdatei wird vom entsprechenden System geladen (vgl. Abschnitt 5.2.4) oder muss vom Crawler-Administrator bereitgestellt werden. Aus dem XML werden anschließend mithilfe von JAXB Java-Objekte erstellt, welche in die zuvor eingeführte ESB-unabhängige Repräsentation im ETG transformiert werden. Statistische Informationen stellt Apache Synapse über JMX bereit, diese stehen also auch als Java-Objekte zur Verfügung. Die Details der Abbildung der internen Datenmodelle von Apache Camel und Apache Synapse auf den ETG sind in [Gru13] dokumentiert.

Die einheitliche Repräsentation von ESBs und deren Routing im ETG ist die Voraussetzung dafür, dass bei der Migration und in anderen Anwendungsfällen die richtigen Entscheidungen getroffen werden. Dennoch ist es nötig, die Semantik der repräsentierten Logik, beispielsweise Transformationen oder Bedingungen, zu verstehen, um bei der Migration von Anwendungen die richtigen Grenzen ziehen und Adaptionen vornehmen zu können. Einerseits kann der Nutzer entscheiden, den ESB sowie die darin registrierten Dienste unverändert zu lassen und nur die auf dem ESB aufbauenden Teile der zusammengesetzten Anwendung zu migrieren. Andererseits kann es sinnvoll sein, auch die Dienste, die durch den ESB angeboten werden, zu migrieren, entweder mitsamt dem ESB oder als direkten Aufruf des Dienstes.

5.2.4 Betriebssystem und Server

Ausgehend von einer Komponente vom Typ *OperatingSystem*, also einem nicht näher spezifizierten Betriebssystem, zeigt dieser Abschnitt, welche Crawler-Plugins während eines Crawler-Laufes ausgeführt werden. Das generische Crawler-Plugin für Betriebssysteme integriert das bestehende Werkzeug „Nmap“ [Lyo14] (vgl. Abschnitt 2.6.5), das mithilfe verschiedener Techniken den Typ (`Linux` in diesem Beispiel) und Hersteller bzw. Derivat (`Ubuntu`) bestimmt. Falls diese Informationen durch ein anderes Plugin von einer darüber liegenden Komponente extrahiert wurde, kann dieses Crawler-Plugin übersprungen werden. Außerdem wird eine Komponente vom Typ

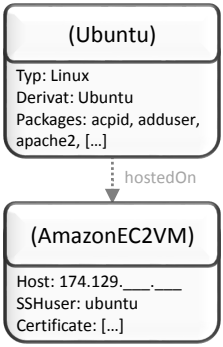

Abbildung 5.6: ETG-Repräsentation eines Betriebssystems und einer VM

VirtualMachine mit der Eigenschaft `Host` angelegt und mit einer Relation vom Typ *hostedOn* mit der Betriebssystemkomponente verbunden, wie in Abbildung 5.6 gezeigt. Das nachfolgende Plugin für virtuelle Maschinen versucht, anhand des IP-Adressbereiches die virtuelle Maschine einem Betreiber zuzuordnen. Im Internet verfügbare Listen geben zum Beispiel an, dass unter anderem der IP-Adressbereich von `174.129.0.0` bis `174.129.255.255` von Amazon EC2[1] und der Bereich von `65.52.128.0` bis `65.52.143.255` von Microsoft Azure[2] genutzt wird. Entsprechend wird der Typ der virtuellen Maschine geändert. Falls für einen Betreiber die VM-Zugangsdaten hinterlegt sind, werden diese in der Komponente als Eigenschaften gesetzt. Mithilfe dieser Zugangsdaten für die virtuelle Maschine kann ein Crawler-Plugin für Ubuntu die genaue Version bestimmen und die installierten Pakete auslesen. Die Liste der Betriebssystem-Pakete ist hilfreich, um eine vergleichbare Linux-Installation in einer anderen Umgebung bereitzustellen. Ausgehend vom Betriebssystem ist es auch möglich, Informationen über die Hardware-Spezifikation zu ermitteln.

[1] „Amazon EC2 Public IP Ranges“:
`https://forums.aws.amazon.com/ann.jspa?annID=1701`

[2] „IP-Bereiche des Azure-Rechenzentrums“:
`http://msdn.microsoft.com/library/azure/dn175718.aspx`

5.3 Entwicklung von Crawler-Plugins

Umso mehr Crawler-Plugins für die verschiedenen Typen von Komponenten verfügbar sind, desto gewinnbringender nutzbar ist der Crawler. Um die Entwicklung bisher nicht vorhandener Crawler-Plugins zu erleichtern, sieht die in Abschnitt 5.1 vorgestellte Crawler-Methode vor, dass Plugins nur die für den Komponententyp spezifische Logik enthalten. Alle weitere Funktionalität wird vom Crawler übernommen und automatisiert. Funktionalitäten, die von mehreren Plugins benötigt werden, stellt der Crawler als wiederverwendbare Dienste in der Architekturkomponente „Plugin-Verwaltung“ zur Verfügung. Durch die Entwicklung weiterer Plugins und die Extraktion neuer wiederverwendbarer Dienste entsteht so ein Rahmenwerk, das sukzessive den Aufwand der Crawler-Plugin-Entwicklung weiter reduziert. Dies ermöglicht es dem Crawler Entwickler, sich auf die Logik zur Extraktion und Verarbeitung der typspezifischen Informationen der Komponente zu fokussieren.

Zur weiteren Reduktion des Aufwands beschreibt dieses Kapitel eine Entwicklungs- und Testmethode für Crawler-Plugins in Abschnitt 5.3.1 und potentielle Informationsquellen in Abschnitt 5.3.2.

5.3.1 Entwicklungs- und Testmethode

Da die Qualität der Crawler-Plugins zentral für die Qualität der resultierenden ETGs ist, stellt dieser Abschnitt eine systematische Entwicklungs- und Testmethode für Crawler-Plugins vor. Nach Ludewig und Lichter [LL10] definiert ein „systematischer Test“ präzise seine Grenzfälle, wählt seine Eingaben systematisch, dokumentiert seine Ergebnisse und evaluiert diese basierend auf Kriterien, die vor dem Test aufgestellt wurden. Der Test der Crawler-Plugins folgt dem von Ludewig und Lichter [LL10] definierten generellen Ablauf für systematische Tests. Durch die Wiederholbarkeit und teilweise Automatisierung sind auch „Regressionstests“ [IEE90] möglich, die nach Änderungen sicherstellen, dass keine ungewollten Seiteneffekte auftreten. Die Entwicklungs- und Testmethode umfasst folgende sechs Phasen, welche auch in Abbildung 5.7 dargestellt werden:

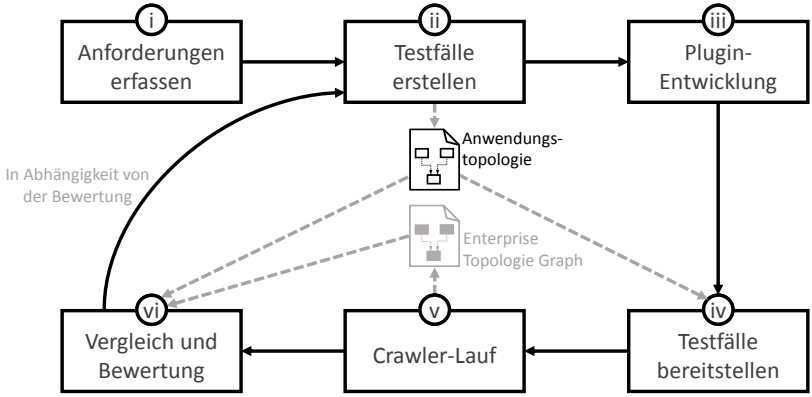

Abbildung 5.7: Entwicklungs- und Testmethode

(i) *Anforderungen erfassen*. Die Erfassung der Anforderungen für Crawler-Plugins wird durch die Definition der Typen von Komponenten und Relationen, die gecrawlt werden sollen, spezifiziert. Typen definieren die Namen und Semantik der Eigenschaften, zum Beispiel in welchem Format die Konfiguration, die Version oder Adresse gespeichert werden soll.

(ii) *Testfälle erstellen*. Passend zu den Anforderungen werden anschließend eine oder mehrere Anwendungen ausgewählt, von denen als Testfall für das entsprechende Plugin ein ETG in einem Crawler-Lauf erstellt werden soll. Die Topologie dieser Anwendung(en) wird in einem maschinenlesbaren Format umgesetzt (vgl. Abschnitt 2.4), um einerseits die Bereitstellung der Anwendung(en) während des Tests zu automatisieren und andererseits am Ende einen automatischen Vergleich zu ermöglichen.

(iii) *Plugin-Entwicklung*. Der eigentliche Entwicklungs- und Testzyklus beginnt in diesem Schritt mit der Entwicklung des Crawler-Plugins basierend auf den Anforderungen von (i).

(iv) *Testfälle bereitstellen*. Vor jedem Testlauf werden die Anwendungen, welche in den Testfällen definiert wurden, automatisiert bereitgestellt.

(v) *Crawler-Lauf*. Ein Crawler-Lauf mit dem zu testenden und gegebenenfalls weiteren Crawler-Plugins wird konfiguriert und durchgeführt. Da-

bei werden Metriken erhoben, wie die Anzahl der aufgerufenen Plugins und deren Laufzeit. Diese Messungen können in Zukunft bei Regressionstests [IEE90] herangezogen werden, um Veränderungen in der Laufzeit von weiterentwickelten Plugins zu identifizieren und unter Umständen Maßnahmen zu ergreifen.

(vi) *Vergleich und Bewertung*. Nachdem der Crawler-Lauf abgeschlossen ist, wird der erstellte ETG, das heißt das Modell der in (ii) bereitgestellten Testfälle, mit der ursprünglichen Anwendungstopologie des Testfalls, das heißt dem Modell, das zur Bereitstellung verwendet wurde, verglichen. Zum Vergleich werden die Metriken Vollständigkeit und Korrektheit verwendet, welche Bestandteil der Anforderung CA-1 („ETG-Qualität“) sind. Dieser Vergleich kann, falls die Maschinenlesbarkeit der Anwendungstopologie gegeben ist, automatisiert werden.

In der hier beschriebenen Entwicklungs- und Testmethode, die den Testfall anfangs explizit definiert, muss ein vollständiger und richtiger ETG erreicht werden. Anderenfalls muss das Crawler-Plugin in (iii) oder der Testfall in (ii) angepasst werden. Nachdem die Funktionalität und die Qualität des Plugins sichergestellt sind, kann es zur Verwendung freigegeben werden.

5.3.2 Informationsquellen für Crawler-Plugins

Die Crawler-Methode schränkt die Art und Weise, wie Crawler-Plugins Informationen ermitteln, nicht ein, wodurch sich eine Vielzahl von potentiellen Informationsquellen ergeben. Dieser Abschnitt diskutiert und klassifiziert verschiedene Informationsquellen mit dem Ziel, Crawler-Plugin-Entwickler zu unterstützen. Die Klassifikation basiert auf der Analyse verschiedener ETGs und den gesammelten Erfahrungen bei der Entwicklung der Crawler-Plugins des Prototyps. Die Liste erhebt keinen Anspruch auf Vollständigkeit, was in einer *offenen Welt* auch nicht möglich wäre.

(i) *Konfiguration*. Praktisch alle wiederverwendbaren Komponenten, beispielsweise Webserver oder DBMS, bieten eine Vielzahl von Konfigurationsmöglichkeiten an, um diese Komponenten an die spezielle Anwendung oder Umgebung anzupassen. Die Konfiguration ist normalerweise strukturiert

und folgt einem bekannten Schema. Dadurch können diese Informationen einfach und zuverlässig extrahiert werden. Informationen über die Konfiguration einer Komponente ermöglichen es, diese Komponente mit verhältnismäßig wenig Aufwand und geringem Transfer von Daten in eine andere Umgebung zu migrieren.[1]

(ii) *Umgebung und Middleware.* Middleware-Komponenten, insbesondere Application Server, Enterprise Service Busse und Message Queueing Systeme, enthalten Informationen über die Konfiguration und Verbindung von Komponenten. Zum Beispiel ermöglicht es ein Java Application Server, Datenbankverbindungen (sogenannte „data sources“) außerhalb der eigentlichen Anwendung zu definieren, so dass diese extrahiert werden können, ohne die Anwendung näher zu betrachten.

(iii) *Implementierung.* Durch die Analyse des Codes einer Komponente ist es möglich, Informationen wie deren Standardkonfiguration, Relationen zu anderen Komponenten, beispielsweise zu einer Datenbank, oder Abhängigkeiten auf die Umgebung zu extrahieren. Diese Analyse ist jedoch aufwendig, da aufgrund der flexiblen Möglichkeiten in Programmiersprachen ein hohes Maß an Programmverständnis nötig ist.

(iv) *Kommunikation zwischen Komponenten.* Die Kommunikation von Komponenten lässt Rückschlüsse auf das Bestehen einer Relation und deren Art zu. Dazu wird die Kommunikation der Komponenten zur Realisierung der Anwendungsfunktionalität mithilfe von Informationen aus dem Betriebssystem analysiert. Für einen Betriebssystemprozess (der von einer ETG Komponente repräsentiert wird) kann so bestimmt werden, auf welchen Netzwerkports und zu welchen Partnern dieser ausgehende Verbindungen öffnet oder ob dieser auf eingehende Verbindungen hört. Falls diese Kommunikation verschlüsselt ist, kann zumindest festgestellt werden, dass eine

[1] Da die Implementierung der Komponente in der spezifischen Version immer gleich ist, kann diese aus einer beliebigen Quelle bezogen werden, zum Beispiel aus dem Internet geladen werden. Anschließend wird die Konfiguration aus der Ursprungsumgebung angewendet, zum Beispiel durch Kopieren der entsprechenden Konfigurationsdatei. Falls die Komponente nicht von der Umgebung abhängig ist, erhält man dadurch eine sich gleich verhaltende Komponente. Die Übertragung der Konfiguration auf eine andere Version der Komponente ist nicht immer möglich und muss im Einzelfall untersucht werden.

Relation besteht. Falls nicht, ist es sogar möglich, durch eine Analyse der übertragenen Daten weitere Informationen zu extrahieren, beispielsweise ob es eine Datenbankverbindung ist und welche Daten verwendet werden. Verwandte Arbeiten, die Informationen aus der Kommunikation von Komponenten extrahieren, werden in Abschnitt 2.6.3 diskutiert.

(v) *Kommunikation mit der Komponente.* Oft ist es möglich, dass Crawler-Plugins direkt mit der Komponente kommunizieren, genauso wie es andere Komponenten oder Benutzer tun würden. Zum Beispiel kann eine Seite des Webservers aufgerufen werden, um aus dem HTTP-Header der Antwort Informationen zu extrahieren. Teilweise gibt es auch dedizierte Statusseiten oder nützliche Informationen können aus durch fehlerhafte Anfragen ausgelösten Fehlermeldungen extrahiert werden.

(vi) *Monitoring und Auditing.* In Geschäftsanwendungen wird eine Vielzahl von technischen Monitoring-Informationen und teilweise auch Auditing-Informationen zur Überprüfung der Regelkonformität gesammelt. Diese Log-Einträge, Ereignisse, Fehlermeldungen usw. sind aktuell und maschinell analysierbar. Zum Beispiel lassen sich die Komponenten, die einen Dienst aufrufen, aus der Log-Datei mit den Netzwerkzugriffen ermitteln.

(vii) *IT-Management-Lösungen*. Aus den in Abschnitt 2.5.3 diskutierten IT-Management-Lösungen können technische Informationen, wie beispielsweise die Konfiguration installierter Software, Lizenzen oder Zugangsdaten von Komponenten, ausgelesen werden. Dabei muss auf die Aktualität der Informationen geachtet werden, da diese Systeme nur einige Aspekte der IT abbilden und nicht zwangsläufig auf dem aktuellen Stand sind.

(viii) *Dokumentation.* Bei der Entwicklung einer Anwendung entstehen verschiedene Dokumente, zum Beispiel die Architektur, ein Deployment-Diagramm oder die Spezifikation. Die Schwäche dieser Informationsquellen ist, dass sie unvollständig sind, schnell veralten und oft nicht nachgezogen werden. Das heißt, dass sich die Anwendung in der Realität und teilweise auch schon während der Entwicklung von der Dokumentation entfernt. Außerdem sind diese Dokumente meist nicht automatisiert auswertbar.

5.4 Evaluation und Validierung

Dieser Abschnitt evaluiert und validiert die in diesem Kapitel vorgestellte Crawler-Methode. Nach der Validierung in Abschnitt 5.4.1 diskutiert Abschnitt 5.4.2 die für den Einsatz des Crawlers benötigten Zugriffsrechte und Sicherheitsaspekte im Allgemeinen. Anschließend wird die Crawler-Methode in den Abschnitten 5.4.3 – 5.4.6 anhand der in Abschnitt 5.1.1 eingeführten Anforderungen evaluiert.

5.4.1 Validierung der Crawler-Methode

Die Crawler-Methode wurde mithilfe der in Abschnitt 7.3 beschrieben prototypischen Implementierung validiert: Einerseits geschah dies anhand der drei in Kapitel 8 geschilderten Fallstudien, andererseits mit weiteren exemplarischen Testanwendungen, welche im Folgenden vorgestellt werden:

(i) „SugarCRM“ [Sug14] ist ein Open-Source Customer Relationship Management System, das vom Standardisierungsgremium des TOSCA-Standards bei OASIS als Referenz-Anwendungsfall und zur Demonstration der Interoperabilität von TOSCA verwendet wird. Die entsprechende CSAR wurde mithilfe einer TOSCA-Laufzeitumgebung bereitgestellt und mit dem Crawler als ETG erstellt. Anschließend wurde durch das Vergleichen des ETGs mit der TOSCA-Anwendungstopologie die Vollständigkeit und Korrektheit des erstellten ETGs validiert, mit dem Ergebnis, dass der ETG alle Informationen enthält, die für eine Migration in eine andere Umgebung benötigt werden.

(ii) „Wordpress“ [Wor14], die am meisten verwendete Blog-Software [Bui14], wurde mithilfe von Amazon CloudFormation [Ama14a] auf Amazon EC2 bereitgestellt, gecrawlt und validiert. Der daraus resultierende ETG enthielt alle Komponenten, Relationen und Eigenschaften, die dafür nötig wären, die AROMA-Methode zu durchlaufen und Wordpress als TOSCA-Anwendungstopologie in einer anderen Umgebung bereitzustellen.

(iii) Eine Java-basierte Testanwendung, die Teile des TOSCA-Ökosystems OpenTOSCA [BBH+13] umfasst, wurde mit Chef [Che14a] sowohl bei Amazon EC2 als auch bei Microsoft Azure bereitgestellt und in beiden Clouds

entsprechend gecrawlt. Die beiden resultierenden ETGs waren, bis auf die unterschiedlichen Infrastruktur-Dienste, gleich. Auch bei dieser Testanwendung waren alle für eine Migration nötigen Informationen im ETG vorhanden.

(iv) Zum Test des ETG-Crawlers bei der Erstellung großer ETGs wurde eine Testanwendung mit vielen Komponenten generiert, die mit Chef [Che14a] bei Amazon EC2 bereitgestellt wurde. Der davon durch den ETG-Crawler erstellte ETG hatte 1060 Komponenten und konnte durch die im Rahmen der vorliegenden Arbeit entwickelten Prototypen verarbeitet werden.

Diese exemplarischen Testanwendungen zeigen, neben den in Kapitel 8 beschriebenen Fallstudien, die Validität und Anwendbarkeit des ETG-Crawlers auf verschiedene Arten von Anwendungen und Komponenten.

5.4.2 Zugriffsrechte und Sicherheit

Um Informationen aus den in Abschnitt 5.3.2 vorgestellten Informationsquellen verarbeiten zu können, müssen Crawler-Plugins diese abfragen, wozu meistens eine Authentifizierung nötig ist. Die Daten, welche zur Authentifizierung nötig sind, können durch ein entsprechendes Crawler-Plugin gesetzt, durch ein Single-Sign-on-System bereitgestellt oder vom Crawler-Administrator erfragt werden.

Viele Crawler-Plugins benötigen Zugriff auf das darunterliegende Betriebssystem, um beispielsweise Dateien oder andere Informationen zu extrahieren. Auch für andere Informationsquellen, wie die Netzwerk- oder Speicherkonfiguration, werden Zugriffsrechte benötigt, die normalerweise nur eine kleine Anzahl von Administratoren besitzen. Insgesamt brauchen die Plugins viele Zugriffsrechte in der Produktionsumgebung, was aus organisatorischer Sicht und aufgrund von Sicherheits- und Compliance-Vorgaben in der Praxis zu Widerständen führen kann. Dem kann, neben der entsprechenden Managemententscheidung, begegnet werden, indem den Crawler-Plugins nur Zugangsdaten mit Leseberechtigung gegeben werden.

Ein verwandtes Problem ist, dass in Firmen aus Sicherheits- oder Verwaltungsgründen oft isolierte Netzwerksegmente existieren. Eine Lösung dafür ist, jedes Netzwerksegment separat zu crawlen und die verschiedenen Teile am Ende an einer zentralen Stelle zusammenzuführen.

5.4.3 Erweiterbarkeit und Integration bestehender Werkzeuge

Die Crawler-Methode schränkt die möglichen Typen von Komponenten und Relationen nicht ein und stellt nur geringe Anforderungen an Crawler-Plugins. Beliebige Plugins können sich für beliebige Komponententypen unter Angabe ihrer Abhängigkeiten auf den ETG (vgl. Definition 5.2) registrieren. Die Erweiterung des Crawlers durch die Untersützung neuer Komponententypen kann durch die Registrierung der entsprechenden Crawler-Plugins erreicht werden. Plugins sind so von anderen Plugins, dem ETG und Architekturkomponenten abgeschirmt, dass sie diese nicht negativ beeinflussen können und dass alle Änderungen am ETG festgestellt und geprüft werden können.

Die Implementierung eines Crawler-Plugins kann alles tun, um Informationen zu extrahieren, beispielsweise Webservices, Skripte oder Betriebssystemfunktionalitäten aufrufen. Somit ist auch die Integration bestehender Werkzeuge möglich (Anforderung CA-3), was durch die Integration von „Nmap“ und „Httprint“ in Crawler-Plugins gezeigt wurde (vgl. Abschnitt 5.2.2 bzw. Abschnitt 5.2.4). Die Deduplizierung (Abschnitt 5.1.6) stellt sicher, dass trotz verschiedener Crawler-Plugins, die den ETG gemeinsam erstellen, ein qualitativ hochwertiger ETG entsteht. Die Komponenten der Architektur für die Crawler-Methode sind lose gekoppelt und haben klare Aufgaben und Schnittstellen, so dass einzelne Komponenten weitestgehend unabhängig voneinander verbessert oder ersetzt werden können (vgl. Abschnitt 5.1.3).

5.4.4 Aktualisierung von ETGs

Der in Abschnitt 5.1.5.5 beschriebene Mechanismus der Crawler-Methode adressiert Anforderung CA-4 („Aktualisierung von ETGs“). Aufgrund der Anforderung CA-5, die eine Änderung der IT, zum Beispiel um bei Änderungen den Crawler zu informieren, nicht zulässt, ist es erforderlich, die Aktualisierung manuell anzustoßen oder regelmäßig automatisch ausführen zu lassen. Zur Optimierung der Kosten für das Crawling von ETGs und zur Minimierung des Einflusses auf Produktionssysteme ist eine durchgehen-

de Aktualisierung von ETGs nicht sinnvoll. Dies führt dazu, dass ein ETG nicht immer aktuell ist und somit die Qualität des ETGs (Anforderung CA-1) sinkt, insbesondere dessen Aktualität. Indem vor größeren Adaptionen, wie beispielsweise einer Migration, eine Aktualisierung des ETGs durchgeführt wird, stellt dies jedoch keine Einschränkung dar. Bei Anwendungsfällen, die ständige Aktualität verlangen, wie beispielsweise im Bereich Compliance oder Monitoring, ist dies gesondert zu beachten.

5.4.5 Einfluss auf Produktionssysteme minimieren

Die Minimierung des Einflusses auf Produktionssysteme (Anforderung CA-5) ist für die Akzeptanz und Anwendbarkeit von großer Bedeutung und wird daher durch verschiedene Maßnahmen adressiert: Die vorgestellte Crawler-Methode verlangt keine Veränderung der Produktionssysteme, beispielsweise die Installation von Agenten. Die Crawler-Plugins lesen ausschließlich Informationen von den Produktionssystemen, was durch entsprechende Einschränkungen der Zugriffsrechte sichergestellt werden kann. Die von Crawler-Plugins extrahierten Rohdaten werden auf den Servern, welche die Crawler-Plugins betreiben, weiterverarbeitet, womit der Großteil der benötigten Ressourcen außerhalb des Produktionssystems verbraucht wird. Extrahierte Artefakte werden zwischengespeichert (vgl. Abschnitt 5.1.3), damit jedes Artefakt nur einmal geladen wird, auch wenn es von verschiedenen Plugins analysiert wird, die sich nicht kennen. Die in Abschnitt 5.1.5 vorgestellte Ablaufsteuerung gewährleistet, dass ausschließlich Crawler-Plugins ausgeführt werden, die neue Informationen extrahieren könnten. Das wird erreicht, indem ein Plugin nur wiederholt auf einer Komponente ausgeführt wird, falls sich seit der letzten Ausführung für das Plugin relevante Teile des ETGs geändert haben, also die Komponente selbst oder die anderen Abhängigkeiten zum ETG. In anderen Worten terminiert ein Crawler-Lauf, wenn kein Crawler-Plugin auf keiner Komponente weitere Informationen ermitteln kann.

5.4.6 ETG-Qualität

Die ETG-Qualität, welche Gegenstand von Anforderung CA-1 ist, teilt sich in die vier Kriterien (i) Aktualität, (ii) Vollständigkeit, (iii) Korrektheit und (iv) richtige Granularität auf. Alle vier Kriterien sind signifikant für die Nutzung des ETGs bei der Anwendungsmigration sowie anderen Anwendungsfällen und werden im Folgenden diskutiert: Die Aktualität des ETGs wird, wie in Abschnitt 5.4.4 erörtert, durch die Möglichkeit adressiert, den ETG zu aktualisieren. Dabei kann durch die Durchführung einer Aktualisierung direkt vor einer Migration die Aktualität des ETGs maximiert werden.

Die Vollständigkeit und Korrektheit eines ETGs kann nicht bewiesen oder durch eine Funktion der Crawler-Methode allgemein gewährleistet werden. Allerdings kann die IT einer Organisation vollständig als ETG repräsentiert werden, wenn für jeden Komponententyp die entsprechenden Crawler-Plugins verfügbar sind. Damit sind die Vollständigkeit und die Korrektheit abhängig von der Qualität der Crawler-Plugins und müssen für jedes Plugin betrachtet werden. Um die Qualität der Crawler-Plugins zu unterstützen, beschreibt die vorliegende Arbeit in Abschnitt 5.3.1 eine Entwicklungs- und Testmethode für Crawler-Plugins. Darüber hinaus sorgt die Crawler-Methode indirekt dafür, den negativen Einfluss von Crawler-Plugins auf die ETG-Qualität zu reduzieren, beispielsweise durch ihre Isolation voneinander (vgl. Abschnitt 5.1.2) und die Kontrolle aller von Crawler-Plugins auf dem ETG ausgeführten Operationen (vgl. Abschnitt 5.1.6).

Ein weiterer Aspekt der Vollständigkeit ist, dass die Crawler-Methode nur Komponenten und Relationen findet, die zum Zeitpunkt des Crawler-Laufes erreichbar sind. Falls zum Beispiel eine virtuelle Maschine nachts herunter gefahren und der Crawler-Lauf dann durchgeführt wird, kann diese nicht gefunden werden. Gegenüber rein auf dem Netzwerk basierenden Ansätzen (vgl. Abschnitt 2.6.3) ist die vorgestellte Methode jedoch überlegen, da meist kein Netzwerkverkehr benötigt wird, um eine Relation zu finden.

Die Granularität der ETGs wird über die Definition der Typen von Komponenten und Relationen bestimmt (siehe Abschnitt 4.3). Crawler-Plugins müssen sich an diesen Typen orientieren, weil sie Komponenten und Relatio-

nen dieser Typen anlegen und die Eigenschaften entsprechend setzen. Somit kann durch die Definition von Typen und die Auswahl der Crawler-Plugins eine abgestimmte und dem Anwendungsfall entsprechende Granularität des ETGs erreicht werden.

5.5 Diskussion und Zusammenfassung

Dieses Kapitel stellt eine Methode zum automatisierten Crawling der in Kapitel 4 eingeführten Enterprise Topologie Graphen vor. ETGs erlauben einen technisch detaillierten Einblick in die IT, mit allen Komponenten und deren Relationen, und erleichtern es, diese zu adaptieren, zu analysieren und zu optimieren. Dabei deckt die vorgestellte Crawler-Methode alle nötigen Schritte ab, beginnend von der Entwicklung und dem Test der Crawler-Plugins, über deren Ausführung in einem Crawler-Lauf, bis hin zum Stellen von Rückfragen an den Nutzer und die Sicherstellung der Qualität des resultierenden ETGs. Neben den in Abschnitt 5.1.1 eingeführten Anforderungen war die Automatisierung von zentraler Bedeutung, da die manuelle Modellierung, welche bisher zur Erstellung von IT-Instanzmodellen nötig war, zeitaufwändig und fehleranfällig ist. Durch das automatisierte Crawling von ETGs wird eine Automatisierung der Migration von Anwendungen erst ermöglicht. Aber auch in anderen Forschungsbereichen im Kontext der Analyse, Adaption und Optimierung der IT können ETGs in Zukunft gewinnbringend angewandt werden, insbesondere da diese automatisiert und damit kostengünstig erstellt und aktuell gehalten werden können. Durch die Flexibilität bei der Erstellung von Crawler-Plugins müssen diese nicht neu entwickelt werden, sondern können auch auf existierenden Werkzeugen aufbauen, die sich auf die Extraktion bestimmter Informationen spezialisiert haben, wie in Abschnitt 5.2 gezeigt wurde. Außerdem ist es durch die Kapselung von Crawler-Plugins vorstellbar, diese in Zukunft über Marktplätze oder Communitys auszutauschen oder als Dienst anzubieten.

KAPITEL 6

UMSETZUNG DER AROMA-METHODE MIT TOSCA

Dieses Kapitel stellt ein Konzept zur Umsetzung und Automatisierung der AROMA-Methode mit dem in Abschnitt 2.4.3 eingeführten Standard TOSCA vor (Beitrag 5). Es umfasst die Transformation, Adaption, Evaluation, Paketierung und Bereitstellung der zu migrierenden Anwendung in die Cloud und andere durch TOSCA unterstützte Zielumgebungen. Mit der Diskussion des verwendeten Typsystems in Abschnitt 6.1 und des OpenTOSCA-Ökosystems in Abschnitt 6.2 wird die Grundlage für die Umsetzung der AROMA-Methode mit TOSCA gelegt. Die Umsetzung der einzelnen Schritte der AROMA-Methode wird in Abschnitt 6.3 behandelt.

6.1 Typsystem für ETG und TOSCA

Für die Umsetzung der AROMA-Methode mit TOSCA verwendet diese Arbeit konsequenterweise die gleichen Typen im ETG wie auch in TOSCA. Bei der Definition der Typen wurde deshalb der TOSCA Primers [OAS13a] als Orientierung verwendet. Dieser definiert teilweise schon Typen, insbesondere die Wurzeln der Vererbungshierarchie, und sieht die flexible Erweiterung der Typen vor. Da das verwendete Typsystem die Granularität des resultierenden ETG bestimmt (vgl. Abschnitt 4.3), ist sichergestellt, dass am Ende der Migration die resultierende Anwendungstopologie in einer Granularität vorliegt, die dessen automatisierte Bereitstellung mit TOSCA erlaubt. Aufgrund der Erweiterbarkeit des TOSCA-Typsystems und da keine Anpassungen an der Methode für dessen Verwendung nötig sind, ist dies keine Einschränkung der Allgemeinheit.

Zur Veranschaulichung wird im Folgenden ein kleiner Ausschnitt der Namen der in den Prototypen verwendeten Typen gezeigt:

- Application
  - JavaApplication
  - PHPApplication
- Database
  - MySQLDatabase
- OperatingSystem
  - Linux
  - Windows
- RDBMS
  - MySQLRDBMS
- VirtualMachine
  - AmazonEC2VM
  - AzureVM
  - OpenStackVM
- Webserver
  - ApacheWebserver
  - GoogleAppEngine
  - TomcatWebserver

6.2 TOSCA-Ökosystem „OpenTOSCA“

Verschiedene Organisationen arbeiten an der Umsetzung eines auf dem TOSCA-Standard [OAS13b] basierenden Ökosystems oder Teilen davon.[1] Das für den Prototyp der vorliegenden Arbeit verwendete TOSCA-Ökosystem ist „OpenTOSCA“ [BBKL14d], das unter der Apache 2.0 Lizenz [Apa04] als Open-Source-Software veröffentlicht wurde. Wie in Abbildung 6.1 gezeigt, setzt sich OpenTOSCA aus dem grafischen Modellierungswerkzeug „Winery“[2] [KBBL13], der Laufzeitumgebung „OpenTOSCA Container“ [BBH+13] und dem Selbstbedienungsportal „Vinothek“ [BBKL14e] zusammen.

Wie in Abschnitt 2.4.3 beschrieben wurde, unterstützt TOSCA die imperative und deklarative Verarbeitung von CSARs und den darin enthaltenen Anwendungstopologien. Da die vorgestellte Methode nur die Topologie der Anwendung für die Zielumgebung erstellt, müssen entweder die entsprechenden TOSCA-Pläne manuell erstellt bzw. automatisiert generiert werden oder es muss eine deklarative Laufzeitumgebung verwendet werden. Dazu existiert beispielsweise eine in OpenTOSCA integrierte Lösung [BBK+14a].

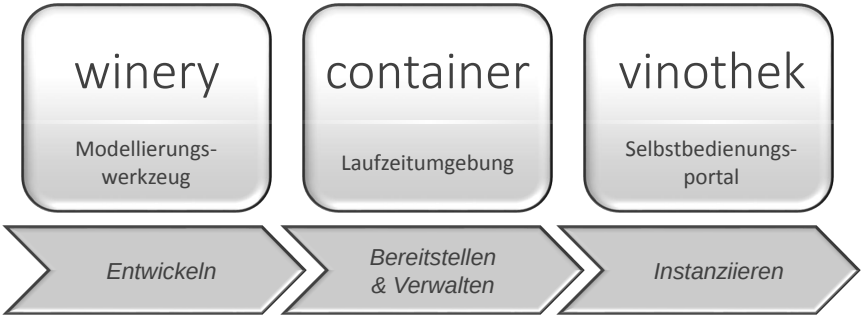

Abbildung 6.1: Übersicht des OpenTOSCA-Ökosystems (übersetzt und modifiziert aus [BBKL14d])

[1] An einer von OASIS organisierten „TOSCA Interoperability Demonstration“ im Rahmen des International Cloud Symposiums nahmen 2013 unter anderen Fujitsu, HP, Huawei, IBM und SAP teil (`https://www.oasis-open.org/events/cloud/2013/TOSCAdemo`).

[2] Winery implementiert die, auch in dieser Arbeit verwendete, visuelle Notation für TOSCA „Vino4TOSCA“ [BBK+12]. Winery wird mittlerweile unter dem Dach der Eclipse Foundation weiterentwickelt (`http://www.eclipse.org/winery/`).

6.3 Umsetzung der Schritte der AROMA-Methode mit TOSCA

Dieser Abschnitt beschäftigt sich mit den Details der Umsetzung der einzelnen Schritte der AROMA-Methode mit TOSCA. Schritt 1 („Crawling der IT“) und Schritt 2 („Identifikation, Partitionierung und Isolation der Anwendung aus der Enterprise Topologie“) der AROMA-Methode arbeiten ausschließlich auf dem ETG und werden nicht durch die Wahl von TOSCA als Sprache zur Beschreibung von Anwendungstopologien beeinflusst. Schritt 1 wird dabei durch die Crawler-Methode aus Kapitel 5 und Schritt 2 durch das in Abschnitt 4.6 vorgestellte Konzept zur Identifikation, Partitionierung und Isolation der zu migrierenden Komponenten automatisiert und realisiert.

6.3.1 Schritt 3: Transformation in Anwendungstopologie

Dieser Abschnitt beschreibt wie eine Menge von ETG-Komponenten, das heißt ein ETG-Segment, in eine TOSCA-Anwendungstopologie transformiert werden. Neben der eigentlichen Transformation findet auch eine Abstraktion des Instanzmodells einer laufenden Anwendung zu einem Modell der Anwendung (d.h. Anwendungstopologie) statt. Dabei soll nur die Transformation stattfinden und die Anwendung an sich nicht adaptiert werden. Dies geschieht im nachfolgenden Schritt der Methode, bei der Anpassung der Anwendung an die Zielumgebung (vgl. Abschnitt 6.3.2). Die Transformation eines ETG-Segments in eine TOSCA-Anwendungstopologie wird in vier Phasen durchgeführt:

Phase 1: Für jede ETG-Komponente und -Relation des ETG-Segments wird im leeren TOSCA-Modell ein NodeTemplate bzw. RelationshipTemplate erstellt. Dabei werden die Ausgangs- und Zielkomponenten der Relationen entsprechend übersetzt, sodass die TOSCA-Anwendungstopologie die gleiche Struktur hat wie das ETG-Segment.

Phase 2: Für jeden Typ der ETG-Komponenten und -Relationen wird ein entsprechender Typ in TOSCA ermittelt und gesetzt. Falls kein gleichnamiger TOSCA-Typ vorhanden ist,[1] wird auf eine globale Tabelle zurückgegriffen,

[1] Dabei wird die eindeutige Typ-ID des ETGs mit dem TOSCA QName, also Namespace und lokaler Name der XML-Definition, verglichen.

welche die ETG-Typen den entsprechenden TOSCA-Typen zuordnet. Diese Tabelle kann einmal, für alle zu diesem Zeitpunkt bekannten Typen, erstellt und anschließend für zukünftige Migrationen wiederverwendet werden.

Phase 3: In dieser Phase werden die Eigenschaften transformiert. Da das Zielmodell TOSCA ist, sind die Werte der in den TOSCA-Typen definierten Eigenschaften zu bestimmen. Die Transformation ist beendet, wenn für jeden in der Definition der Eigenschaften vorgesehenen Wert, welcher keinen *default*-Wert aufweist, ein Wert aus dem ETG bestimmt wurde. Dazu wird sowohl ein generischer Ansatz als auch eine, bei Bedarf darüber hinausgehende, typspezifische Definition vorgesehen: Die automatisierte und generische Zuordnung trifft die Einschränkung, dass die TOSCA-Typen mit XSD definiert wurden sowie ausschließlich flache Listen von Schlüssel-Wert-Paaren als Eigenschaften definieren. Basierend darauf wird versucht, in den TOSCA-Typen definierte Schlüssel, dies entspricht dem Namen des XML-Elements, als Name einer ETG-Eigenschaft zu finden. Gefundene Werte werden in das XML-Dokument kopiert und möglicherweise die in der XSD definierten *default*-Werte überschrieben. Über den generischen Ansatz hinaus kann pro ETG-TOSCA-Typ-Paar eine Zuordnung der Namen der Eigenschaften definiert werden. Diese können in der globalen Tabelle zur Zuordnung der Typen, welche in der vorhergehenden Phase beschrieben wurde, abgelegt werden. Für komplexere Zuordnungen, die zum Beispiel eine Transformation der Werte, Unterstützung von komplexen XML-Datentypen oder die dynamische Bestimmung der Namen der ETG-Eigenschaften vorsehen, kann eine typspezifische Logik implementiert und angewendet werden. Dadurch, dass sich das Typsystem des ETGs an TOSCA orientiert (vgl. Abschnitt 6.1), ist im Normalfall die generische Zuordnung ausreichend.

Phase 4: In der letzten Phase werden die DeploymentArtifacts aus den Artefakten, die den ETG-Komponenten angehängt wurden, erstellt, zum Beispiel eine Konfigurationsdatei oder ein Java Web Archive. Dies kann automatisiert durch eine Heuristik geschehen, welche den Typ des DeploymentArtifacts aus der Dateiendung bestimmt.

Die Erweiterung der Transformation eines ETG-Segments in eine TOSCA-Anwendungstopologie um neue Komponenten- und Relationentypen ist möglich und erfordert nur für manche Typen zusätzliche Maßnahmen. Falls eine automatische Abbildung des neuen Typs auf einen bestehenden TOSCA-Typ nicht möglich ist, müssen die neuen ETG-Typen in die in Phase 2 eingeführte Tabelle eingetragen werden. Falls die generische Transformation der Eigenschaften in Phase 3 nicht anwendbar ist, muss darüber hinaus die explizite Zuordnung der Eigenschaften definiert oder deren Berechnung implementiert werden.

6.3.2 Schritt 4: Adaption an die konkrete Zielumgebung

Dieser Abschnitt zeigt, wie eine Anwendungstopologie so adaptiert wird, dass sie in der vom Benutzer gewählten Zielumgebung betrieben werden kann. Dazu werden, wo nötig, Komponenten ersetzt, zum Beispiel indem eine virtuelle Maschine nach der Adaption von einem Infrastrukturdienst der Zielumgebung bereitgestellt wird oder ein Application Server durch einen entsprechenden Plattformdienst ersetzt wird. Dabei muss sichergestellt werden, dass die Funktionalität der Anwendung erhalten bleibt. Andererseits soll die Anwendung auch möglichst stark von den Eigenschaften der Zielumgebung profitieren.

Um dies zu ermöglichen beschreibt dieser Abschnitt ein Konzept zur automatisierten Anpassung der Anwendung an die Zielumgebung durch die entsprechende Adaption der Anwendungstopologie (TOSCA TopologyTemplate). Dazu wird vom Benutzer eine konkrete Zielumgebung definiert, indem er Anbieter und/oder deren Dienste wählt. Die Definition von Anbietern und Diensten wird in Abschnitt 6.3.2.1 behandelt. Die Durchführung der Adaption der Anwendungstopologie betrachtet Abschnitt 6.3.2.2. Die vorgestellte Lösung beschränkt sich auf die Adaption an die Zielumgebung, das heißt, dass die Anwendung dort lauffähig ist, und betrachtet nicht deren Optimierung (vgl. Abschnitt 2.7). Abschnitt 6.3.2.3 führt verschiedene Strategien für die Bewertung der adaptierten Anwendungstopologien ein. Abschnitt 6.3.2.4

diskutiert das präsentierte Konzept und stellt zwei weitergehende Ansätze kurz vor, die komplexere Adaptionen zulassen.

6.3.2.1 Definition von Zielumgebungen

Vor der ersten Adaption müssen die Betreiber des Systems, das die AROMA-Methode umsetzt, die verfügbaren Anbieter und Dienste definieren, aus denen der Benutzer im Folgenden wählen kann. Zusätzlich könnten die Definitionen über eine zentrale Datenbank ausgetauscht werden oder der Benutzer könnte eigene Anbieter und Dienste definieren. In dem vorgesehenen Modell bietet jeder Anbieter eine Menge von Diensten an. Jeder Dienst wiederum definiert einen oder mehrere TOSCA NodeTypes, welche dieser ersetzen kann, ohne dass weitere Komponenten für deren Betrieb in der Anwendungstopologie nötig wären. Der hier vorgestellte Ansatz erfordert, dass die NodeTemplates, die diese Dienste repräsentieren, keine ausgehenden RelationshipTemplates vom Typ *hostedOn* haben. Der Dienst *Amazon Beanstalk* kann beispielsweise NodeTemplates vom Typ *Apache Tomcat* ersetzen. Optional kann durch die Definition eines *Validators* und *Transformators* die generische Adaption mit dienstspezifischer Logik überschrieben werden. Ein Validator prüft, ob ein Dienst das aktuell bearbeitete NodeTemplate betreiben kann, zum Beispiel ob ein Apache Tomcat mit Version `6.x` durch einen Dienst, der auf Version `7.x` basiert, betrieben werden kann oder ob die von einer Anwendung benötigten Erweiterungen von einem Application Server unterstützt werden. Ein *Transformator* führt die nötigen Adaptionen für die Nutzung eines Dienstes in der Anwendungstopologie durch, zum Beispiel die Umwandlung der Namen der Pakete beim Wechsel der Linux-Distribution.

Für die Darstellung in der Benutzeroberfläche sind die Dienste nach den Arten von Anwendungsbestandteilen gruppiert (vgl. Abschnitt 2.2). So kann der Benutzer zum Beispiel spezifizieren, dass er ausschließlich Infrastrukturdienste nutzen will.

6.3.2.2 Durchführung der Adaption

Die Adaption wird in zwei Phasen durchgeführt: Zuerst werden die kompatiblen Dienste für jedes NodeTemplate bestimmt und anschließend alle mit den Diensten möglichen Anwendungstopologien generiert.

Phase 1: Bestimmung kompatibler Dienste. Für jedes NodeTemplate ohne Anwendungsfunktionalität werden die kompatiblen Dienste ermittelt. Das heißt, es werden alle vom Benutzer gewählten Dienste bestimmt, die den NodeType des aktuell bearbeiteten NodeTemplates ersetzen können. Dies geschieht im Normalfall generisch, indem geprüft wird, ob der NodeType des Dienstes gleich oder spezifischer als der NodeType des NodeTemplates ist (d.h. ein Kindtyp ist). Falls für den Dienst ein spezifischer Validator definiert wurde, wird dieser verwendet, um die Kompatibilität des Dienstes zu prüfen, beispielsweise unter Einbeziehung der Eigenschaften der Komponente.

Phase 2: Generierung möglicher Anwendungstopologien. In der nächsten Phase werden alle Anwendungstopologien generiert, welche sich durch die verschiedenen Kombinationen der kompatiblen Dienste aus Phase 1 ergeben. Die Adaptionen werden umgesetzt, indem der NodeType des NodeTemplates verändert wird. Zum Beispiel könnte der NodeType einer generischen virtuellen Maschine durch *AmazonEC2VM* ersetzt werden, also durch einen dienstspezifischen Typ. Dies ist möglich, da die Vererbung eine Verfeinerung des NodeTypes ist. Hinzugekommene Eigenschaften müssen gegebenenfalls im nächsten Schritt der AROMA-Methode durch den Benutzer ausgefüllt werden. Falls vorhanden wird stattdessen der dienstspezifische Transformator genutzt, der explizit implementiert, welche Änderungen an der Anwendungstopologie vorgenommen werden sollen, beispielsweise um Werte von Eigenschaften zu berechnen. Nachdem das NodeTemplate angepasst wurde, können alle Nodes und Relationships gelöscht werden, die nur von schon adaptierten NodeTemplates erreichbar sind. Die Adaption ist abgeschlossen wenn jedes NodeTemplate der Anwendungstopologie direkt oder indirekt von einem der durch den Benutzer gewählten Dienste betrieben wird.

Anschließend werden die generierten Anwendungstopologien mit den Bewertungsstrategien aus Abschnitt 6.3.2.3 bewertet und die Besten dem Benutzer zur Auswahl angeboten.

6.3.2.3 Bewertungsstrategien

In der Realisierung der vorliegenden Arbeit werden dem Benutzer zwei Bewertungsstrategien angeboten: (i) minimales Risiko und (ii) maximale Nutzung der Zielumgebung.

Die Bewertungsstrategie „Minimales Risiko“ hat das Ziel, alle *nötigen* Änderungen durchzuführen, so dass die Anwendungstopologie in der Zielumgebung lauffähig ist. Dazu werden die Veränderungen minimiert, das heißt, die Anzahl der NodeTemplates und die verwendeten NodeTypes werden möglichst wenig verändert. Gut bewertete Anwendungstopologien bleiben somit nahe an ihrer Struktur in der Ursprungsumgebung, was das Risiko für den Betrieb in der Zielumgebung reduziert.

Die Bewertungsstrategie „Maximale Nutzung der Zielumgebung“ hat das Ziel, die Eigenschaften der Zielumgebung durch die Verwendung ihrer Dienste und ohne Berücksichtigung der ursprünglichen Struktur der Anwendung möglichst stark zu nutzen. Dazu wird die Bewertung darauf ausgerichtet, die Anzahl der NodeTemplates in der resultierenden Anwendungstopologie zu minimieren und somit möglichst viele Teile der Anwendung durch Dienste des Anbieters abzudecken. Durch die stärkere Änderung der Struktur der Anwendung steigt auch das Risiko, da eher Kompatibilitätsprobleme auftreten können.

6.3.2.4 Diskussion und alternative Ansätze

Der vorgestellte Ansatz zeigt, wie bei der Umsetzung der AROMA-Methode mit TOSCA eine Anwendungstopologie an die Zielumgebung angepasst wird. Dabei beschreibt der Benutzer die konkrete Zielumgebung durch die Auswahl von Anbietern oder deren Diensten. Darauf basierend wird die Anwendungstopologie mithilfe der gewählten Dienste umgesetzt. Um die Funktionalität zu erhalten, bleiben die Komponenten, welche die Anwendungsfunktionalität und die Geschäftsprozesse repräsentieren, unverändert. Im Falle der Cloud als Zielumgebung wird, wie in Abschnitt 2.3.2 diskutiert, die Anwendung mindestens eine Cloud-Anwendung im Sinne von Defini-

tion 2.2. Je nach Anwendung liegt nach der Adaption auch eine native Cloud-Anwendung (vgl. Definition 2.3) vor.

Neue Anbieter, Dienste und von den Diensten unterstützte TOSCA NodeTypes können, wie in Abschnitt 6.3.2.1 beschrieben, stets im System hinterlegt werden. Falls für einen Anbieter oder Dienst die generische Logik zur Adaption mit einer dienstspezifischen überschrieben werden soll, müssen dafür die entsprechenden Validatoren und Transformatoren implementiert werden. Außerdem können neue, für weitere Anwendungsfälle angepasste Bewertungsstrategien (Abschnitt 6.3.2.3) gegen die dafür vorgesehene Schnittstelle implementiert werden.

Als Alternative, insbesondere wenn die Bewertungsstrategie (ii) gewählt wird, beschreiben Hirmer et al. [HBBL14] einen Ansatz, der die Anwendungstopologie komplett neu aufbaut. Um diesen Ansatz anzuwenden, werden zuerst alle NodeTemplates aus der Anwendungstopologie entfernt, die keine Anwendungsfunktionalität repräsentieren. Ausgehend von den TOSCA Requirements und Capabilities der NodeTemplates mit Anwendungsfunktionalität wird anschließend die Anwendungstopologie neu aufgebaut. Dieser Ansatz kann die verschiedenen Anwendungstopologien einer Organisation bezüglich der Nutzung und Konfiguration der Komponenten homogenisieren. Dadurch dass die Anwendungstopologie neu aufgebaut wird und möglicherweise eine andere Struktur hat wie die ursprüngliche Topologie, steigt jedoch auch das Risiko, bisher unbekannte Seiteneffekte und Annahmen hinsichtlich der Kombination und Konfiguration der Komponenten nicht zu berücksichtigen.

Das vorgestellte Konzept ist für Adaptionen geeignet, die sich auf einzelne NodeTemplates beziehen. Ein auf Mustern basierender Ansatz für komplexe Adaptionen, die mehrere NodeTemplates umfassen und viele Abhängigkeiten haben, wurde mit *Automated Management Patterns* von Breitenbücher et al. [BBK+14b] vorgestellt.

6.3.3 Schritt 5: Evaluation und manuelle Anpassung

Um dem Benutzer die Evaluation und gegebenenfalls Adaption der TOSCA-Anwendungstopologie zu ermöglichen, wird diese, mitsamt aller Artefakte, in das in Abschnitt 6.2 vorgestellte grafische TOSCA-Modellierungswerkzeug Winery importiert. In Winery ist es möglich, die TOSCA-Anwendungstopologie zu evaluieren und zu bearbeiten. Durch die Nutzung des TOSCA-Standards ist in diesem Schritt auch der Einsatz anderer Werkzeuge möglich, die den TOSCA-Standard unterstützen.

Falls die Bereitstellung mit einer imperativen TOSCA-Laufzeitumgebung (vgl. Abschnitt 2.4.3.2) durchgeführt werden soll, muss in diesem Schritt der entsprechende Plan für die Bereitstellung manuell modelliert und in Winery hinterlegt werden. Auf diese Domäne angepasste Modellierungssprachen für Pläne, wie etwas BPMN4TOSCA [KBBL12], können den Aufwand der Plan-Modellierung reduzieren. Alternativ kann ein Build-Plan auch aus der Anwendungstopologie generiert werden, beispielsweise mit den Arbeiten von Breitenbücher et al. [BBK+14a]. Für die deklarative Bereitstellung ist hingegen die TOSCA-Anwendungstopologie ausreichend.

6.3.4 Schritt 6: Paketierung

Winery wird auch für die Paketierung der evaluierten Anwendungstopologie im TOSCA-Paketformat CSAR genutzt. Dabei wird die Anwendungstopologie sowie alle referenzierten Typdefinitionen als XML serialisiert und zusammen mit den Plänen und Artefakten in eine ausführbare, standardkonforme CSAR gepackt [KBBL13]. Da die TOSCA-Typdefinitionen in Winery bereits vorliegen, enthält die CSAR automatisch auch die Implementierung der auf den NodeTypes und RelationshipTypes definierten Operationen.

6.3.5 Schritt 7: Bereitstellung und Umstellung

Diese CSAR wird dem Benutzer zum Download angeboten und mit der entsprechenden TOSCA-Laufzeitumgebung bereitgestellt. Die Auswahl der Laufzeitumgebung hängt davon ab, ob in Schritt 5 (vgl. Abschnitt 6.3.3) die

deklarative oder die imperative Durchführung der Bereitstellung gewählt wurde. Falls eine deklarative Laufzeitumgebung verwendet wird, muss diese die in der Anwendungstopologie verwendeten NodeTypes und RelationshipTypes unterstützen bzw. für neue Typen entsprechend erweitert werden.

Die Teilschritte Test, Datenmigration, Umstellung und Außerbetriebnahme in der Ursprungsumgebung (Schritt 7) werden nicht im Rahmen der vorliegenden Arbeit betrachtet (vgl. Abschnitt 3.2.7).

KAPITEL 7

ARCHITEKTUR UND PROTOTYPEN

Dieses Kapitel umfasst die Architektur und Realisierung der Prototypen, welche die Forschungsbeiträge der vorliegenden Arbeit implementieren. Die Gesamtarchitektur und das „ETG-Framework“, eine erweiterbare Plattform für Software, welche den ETG nutzt, werden in Abschnitt 7.1 eingeführt. In dieses Framework werden die drei Prototypen „ETG-Verwaltung“, deren Architektur und Realisierung in Abschnitt 7.2 vorgestellt wird, der „ETG-Crawler“ (Abschnitt 7.3) und der „AROMA-Migrationsassistent“ (Abschnitt 7.4) integriert.

7.1 Gesamtarchitektur und ETG-Framework

Wie in Abbildung 7.1 dargestellt, ist die Gesamtarchitektur in drei Schichten strukturiert: (i) Benutzeroberfläche, (ii) Geschäftslogik und (iii) Datenhaltung. Jeder der drei in diesem Kapitel vorgestellten Prototypen umfasst Architekturkomponenten auf allen drei Schichten. Mögliche zukünftige Erweiterungen werden auf der rechten Seite angedeutet.

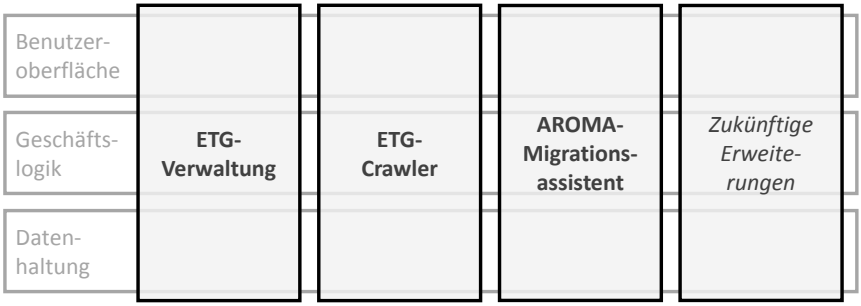

Abbildung 7.1: Gesamtarchitektur

Das ETG-Framework setzt diese Architektur um und realisiert damit eine erweiterbare Plattform, auf der die drei Prototypen der vorliegenden Arbeit aufbauen. Dazu stellt das ETG-Framework ein Rahmenwerk für die Benutzeroberfläche und Geschäftslogik bereit: Die Architekturschicht Geschäftslogik wurde mit der „Java Enterprise Edition“ (JavaEE) [DS13] realisiert. Jede Architekturkomponente der Geschäftslogik wird dabei durch eine zustandslose „Enterprise Java Bean“ (EJB) [Vat13] implementiert. Über das „Java Naming and Directory Interface“ (JNDI) [Ora14] kann jede Architekturkomponente eine Referenz zu jeder anderen Architekturkomponente ermitteln und diese aufrufen. Die Benutzeroberfläche stellt die einheitliche Darstellung und Navigation aller Prototypen sicher. Diese wurde mit „Vaadin“ [Vaa14] realisiert, einem auf dem „Google Web Toolkit“ (GWT) [GWT14] aufbauenden Komponenten-Framework, das es ermöglicht, grafische Benutzeroberflächen in Java zu implementieren. Der Teil der Benutzeroberfläche, der später im Browser des Benutzers läuft, wird nach JavaScript kompiliert und zusammen mit dem Server-Teil als Java Web Archive (WAR) gepackt. Vaadin abstrahiert dabei von den technischen Details der Kommunikation zwischen dem Browser (*Client*) und dem Server, so dass die Benutzeroberfläche im Browser transparent auf die EJBs der Geschäftslogik zugreifen kann. Abbildung 7.2 zeigt die Einstiegsseite des ETG-Frameworks.

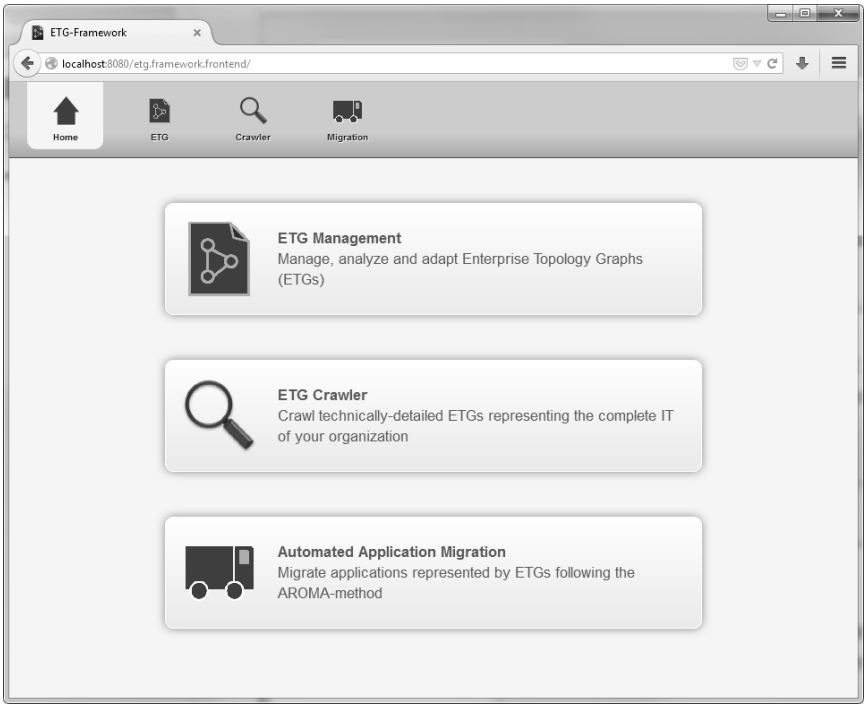

Abbildung 7.2: ETG-Framework

7.2 ETG-Verwaltung

Die ETG-Verwaltung stellt ETGs und die verschiedenen Operationen auf ETGs allen auf dem ETG-Framework aufbauenden Prototypen zur Verfügung. Dabei muss nach Definition 4.1 jeder ETG etg_t zu verschiedenen logischen Zeitpunkten t verwaltet werden, zu dem die IT in dem im ETG repräsentierten Zustand war, bzw. zu dem der Benutzer den ETG in dieser Version gespeichert hat. Dies ermöglicht es zum Beispiel, Veränderungen und Trends der IT nachzuvollziehen und zu analysieren. Die ETG-Verwaltung erlaubt es auch, den ETG zu verändern, die Repräsentation also so zu modifizieren, dass sie sich von der Realität entfernt. Dies kann beispielsweise zur Abstraktion für einen bestimmten Anwendungsfall verwendet werden oder um einen zukünftigen Zustand zu skizzieren. Diese Änderungen können in einem neuen ETG gespeichert und somit auch unabhängig versioniert werden. Alle Ausprägungen und Versionen des ETGs einer Organisation werden in der ETG-Verwaltung zu einem „ETG-Projekt“ zusammengefasst.

Abbildung 7.3 zeigt die Architektur der ETG-Verwaltung. Diese sieht, in Abhängigkeit von den zu speichernden Daten, verschiedene Arten von Datenbankmanagementsystemen für die Datenhaltung vor: Metainforma-

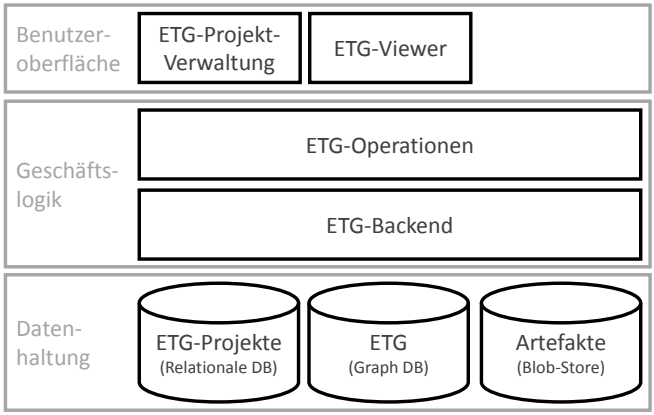

Abbildung 7.3: Architektur der ETG-Verwaltung

tionen für die Verwaltung von ETG-Projekten und ETGs werden in einer relationalen Datenbank gespeichert. Dazu wird im Prototyp „Apache Derby“ [Apa14b], ein vollständig in Java implementiertes RDBMS, verwendet. Die eigentlichen ETGs werden in einer Graph-Datenbank gespeichert, welche auf die Verarbeitung von Graphen zugeschnittene Funktionalitäten anbietet, zum Beispiel Graph-Traversierung. Dazu wird im Prototyp „Neo4J“ [Neo14], eine in Java implementierte Graph-Datenbank, verwendet. Für Artefakte, das heißt verhältnismäßig große Binärdateien wie Konfigurations- oder Programmdateien, wird ein separater Blob-Store eingesetzt. Zur Speicherung der Artefakte wird im Prototyp das Dateisystem des Betriebssystems genutzt, in dem die Artefakte unter einer eindeutigen ID abgelegt werden. Mit dieser ID werden die Artefakte dann vom ETG aus referenziert. Für das Clustering und die Unterstützung vieler und großer Artefakte müsste der Prototyp dahingehend erweitert werden, dass er einen verteilten Blob-Store, wie zum Beispiel Amazon S3 [Ama14c] oder GlusterFW [Red14], verwendet.

In der Architekturschicht Geschäftslogik abstrahiert das „ETG-Backend“ von den verwendeten Technologien zur Datenhaltung und gibt ausschließlich Java-Objekte im jeweiligen Domänenmodell an die anderen Architekturkomponenten zurück, beispielsweise im ETG-Datenmodell anstatt in dem von Neo4j. Neben der Verwaltung von ETGs und ETG-Projekten werden zum Beispiel die in Kapitel 4 vorgestellten Operationen, eine Suchfunktion, sowie der Import und Export des ETGs bereitgestellt.

In der Architekturschicht Benutzeroberfläche ermöglicht die Architekturkomponente „ETG-Projekt-Verwaltung“ das Erstellen, Anzeigen, Bearbeiten, Importieren, Exportieren und Löschen von ETGs und ETG-Projekten. Der „ETG-Viewer“ erlaubt es ETGs zu visualisieren. Abbildung 7.4 zeigt einen Screenshot der Benutzeroberfläche des Prototyps.

Zur Validierung des ETG-Verwaltung hinsichtlich der Verarbeitung großer ETGs wurden Tests zum Importieren, Laden, Anzeigen (seitenweise) und Löschen von ETGs mit über 100.000 Komponenten erfolgreich durchgeführt.

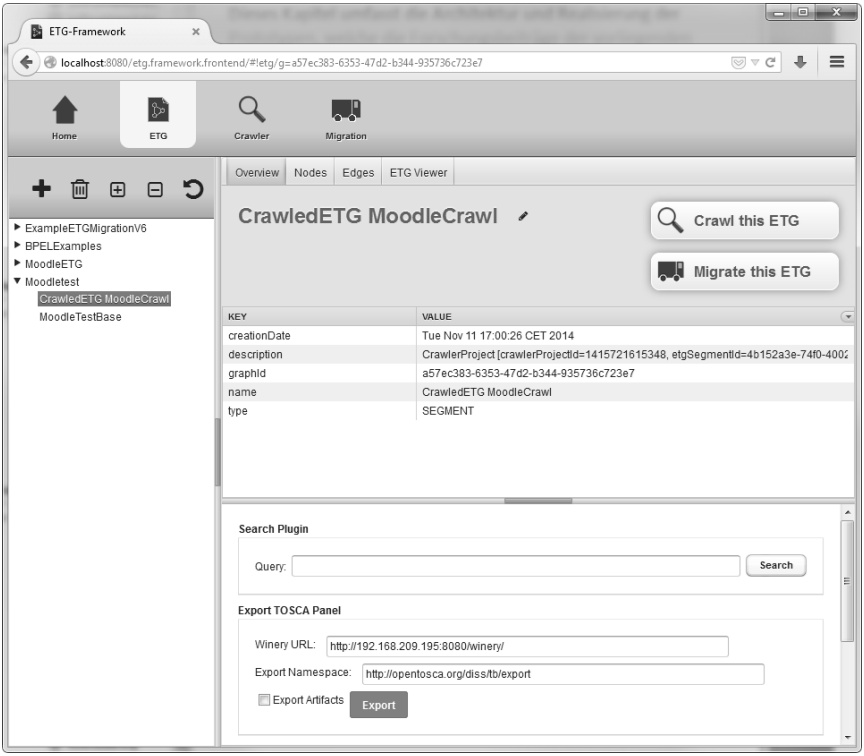

Abbildung 7.4: Screenshot der Benutzeroberfläche – ETG-Verwaltung

7.3 ETG-Crawler

Dieser Abschnitt beschreibt den Prototyp der in Kapitel 5 vorgestellten Crawler-Methode. Dabei beschreibt Abschnitt 7.3.1 die Architektur des Crawlers und Abschnitt 7.3.2 die Realisierung der Architektur. Eine Übersicht aller für den Crawler entwickelten Crawler-Plugins gibt Abschnitt 7.3.3.

7.3.1 Architektur des ETG-Crawlers

Abbildung 7.5 zeigt die Architektur des ETG-Crawlers, welche die abstrakte Architektur der Crawler-Umgebung aus Abschnitt 5.1.3 verfeinert. Die Steuerung von Crawler-Läufen und deren Konfigurationen, zum Beispiel die für einen Crawler-Lauf ausgewählten Plugins, wird von der Architekturkomponente „Crawler-Verwaltung“ und der entsprechenden Benutzeroberfläche abgedeckt. Diese Daten werden in einer relationalen Datenbank abgelegt. Crawler-Plugins registrieren sich bei der „Plugin-Verwaltung“ und werden von dieser der „Ablaufsteuerung“ bereitstellt. Die Ablaufsteuerung entscheidet nach dem Start eines Crawler-Laufes durch die Crawler-Verwaltung, welches Crawler-Plugin wann und auf welcher ETG-Komponente ausge-

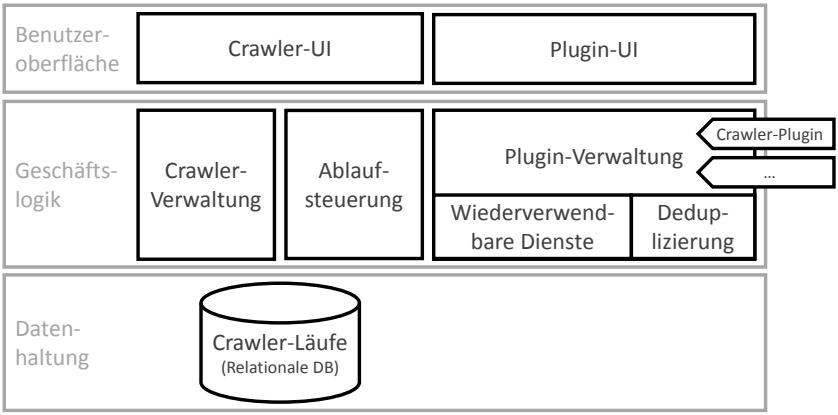

Abbildung 7.5: Architektur des ETG-Crawlers

führt wird. Alle Änderungen von Crawler-Plugins am ETG werden von der Architekturkomponente „Deduplizierung“ geprüft, deren generische Verarbeitungslogik bei Bedarf mit typspezifischen Plugins überschrieben werden kann. Die in Abschnitt 5.3 behandelten „Wiederverwendbaren Dienste“ stellen verschiedene zustandslose Funktionalitäten zur Verfügung, die von verschiedenen Crawler-Plugins verwendet werden können, beispielsweise das Herunterladen von Dateien, das Validieren und Transformieren von Daten oder das Herstellen von Verbindungen (SSH, FTP, ...).

Die gewählte Architektur stellt somit sicher, dass die vom Crawler unterstützten Komponententypen jederzeit einfach erweitert werden können. Crawler-Plugins sind voneinander isoliert, lose gekoppelt und tauschen Daten nur indirekt über den ETG aus (vgl. Abschnitt 5.1.2).

7.3.2 Realisierung des ETG-Crawlers

Als Crawler-Plugin kann jede Java-Klasse genutzt werden, welche die Klasse `AbstractCrawlerPlugin` erweitert. Diese abstrakte Klasse erzwingt, dass jedes Crawler-Plugin eine ID, einen Namen und die Komponententypen angibt, die es bearbeiten kann. Außerdem können Abhängigkeiten, welche über die ETG-Komponente hinausgehen, auf der das Crawler-Plugin ausgeführt wird, durch Topologie Queries angegeben werden. Diese Informationen werden von der Architekturkomponente „Plugin-Verwaltung“ abgefragt, um einerseits der Ablaufsteuerung alle Crawler-Plugins bereitzustellen, die einen bestimmten Komponententyp bearbeiten können, und andererseits die Plugins für Crawler-Läufe auszuwählen.

Beim Aufruf eines Crawler-Plugins übergibt die Ablaufsteuerung die zu bearbeitende ETG-Komponente, den ETG und die Ergebnisse der durch Topologie Queries definierten Abhängigkeiten auf den ETG (vgl. Definition 5.2). Das Crawler-Plugin kann nun alles in Java mögliche tun, um die nötigen Informationen zu extrahieren. Alle an das Crawler-Plugin übergebenen Objekte erlauben jedoch nur lesenden Zugriff auf die jeweiligen Daten. Aufrufe von Operationen, die den ETG ändern, werden über die Architekturkomponente „Deduplizierung“ abgewickelt. Falls eine Komponente, die zum

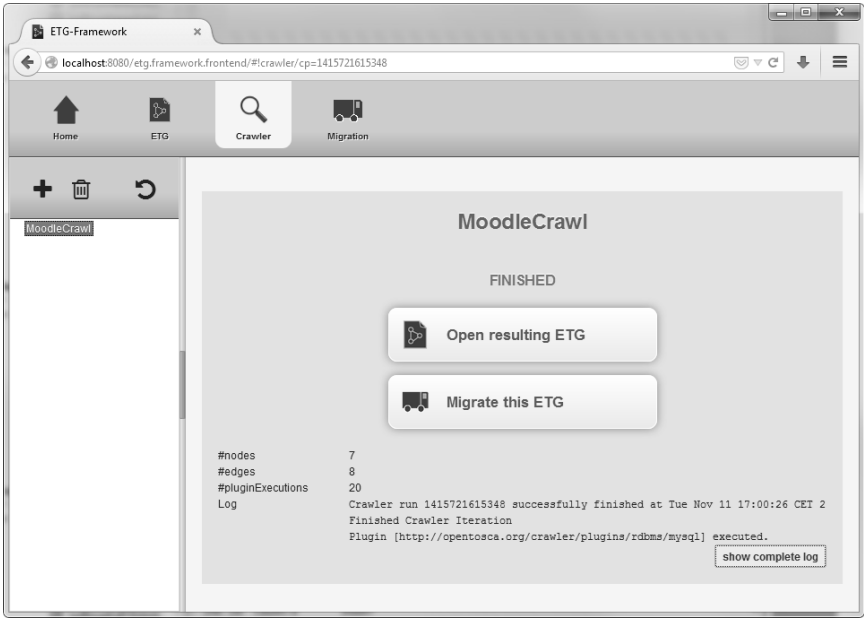

Abbildung 7.6: Benutzeroberfläche Crawler

ETG hinzugefügt werden soll, in diesem bereits existiert, werden die beiden Komponenten zusammengeführt und die existierende Komponente wird an das Crawler-Plugin zur weiteren Verarbeitung zurückgegeben. Die Ablaufsteuerung und die Deduplizierung implementieren die in Abschnitt 5.1.5 und Abschnitt 5.1.6 beschriebenen Algorithmen in Java.

Abbildung 7.6 zeigt die Benutzeroberfläche eines abgeschlossenen Crawler-Laufes im Rahmen der Benutzeroberfläche der Crawler-Verwaltung.

7.3.3 Entwickelte Crawler-Plugins

Zur Validierung der Crawler-Methode wurden 28 prototypische Crawler-Plugins entwickelt, die im Folgenden aufgelistet und kurz erläutert werden. In Abschnitt 5.2 wird zur Veranschaulichung der Crawler-Methode die Funktionsweise einer Auswahl von Crawler-Plugins detailliert beschrieben.

1. ActiveMQJMX – Extrahiert mithilfe von JMX Informationen über den Message Broker Apache Active MQ, wenn eine entsprechende JMS Queue gefunden wurde, zum Beispiel in einer WSDL-Datei.
2. Application – Generisches Crawler-Plugin, das bestimmt, um welche Art von Anwendung es sich handelt und gegebenenfalls von welcher Komponente diese bereitgestellt wird.
3. ApplicationJava – Erweitert den ETG um Angaben, auf welchem Webserver eine Java-Anwendung gehostet wird, falls die entsprechende Komponente eine Eigenschaft `URL` besitzt.
4. ApplicationJavaSSH – Extrahiert die gefundene Java-Anwendung als Artefakt, falls SSH-Zugangsdaten vorhanden sind.
5. ApplicationPHP – Erstellt, falls nicht vorhanden, eine ETG-Komponente für das PHP-Modul sowie den entsprechenden Webserver und versucht diese Komponenten mit Details wie der Version zu ergänzen.
6. ApplicationPHPGenericDB – Analysiert die zur einer PHP-Anwendung gehörenden Dateien, um zu bestimmen, ob diese Anwendung eine Datenbankverbindung herstellt. Ist dies der Fall, so wird versucht, die Informationen, welche für die Herstellung einer Datenbankverbindung nötig sind, generisch zu extrahieren. Dazu werden verschiedene, oft verwendeten Arten zur Konfiguration von PHP-Anwendung (engl. *best practices*) ausprobiert.
7. ApplicationSynapse – Erstellt aus der entsprechenden Konfiguration (XML-Datei) eine Repräsentation der Route im ETG.
8. ApplicationCamel – Erstellt aus den per JMX zugänglichen Informationen eine Repräsentation der Route im ETG.
9. BPEL – Analysiert den Geschäftsprozess und erstellt für jeden aufgerufenen PortType eine entsprechende ETG-Komponente.
10. ESBRoute – Stellt für einen gegebenen ESB-Endpunkt fest, ob dieser durch den ESB Apache Synapse oder Apache Camel realisiert wird, und ändert den Typ der Komponente entsprechend.
11. MySQL – Erstellt, falls nicht vorhanden, eine ETG-Komponente für das Betriebssystem, auf dem das MySQL RDBMS läuft.
12. MySQLSSH – Analysiert die Konfigurationsdateien des MySQL RDBMS,

hängt diese an die ETG-Komponente an und extrahiert eine Auswahl wichtiger Konfigurationsparameter als Eigenschaften der Komponente.

13. MySQLDatabase – Extrahiert die Liste der Tabellen welche in der Datenbank vorhanden sind, speichert diese als Eigenschaft und ändert gegebenenfalls den Typ der Datenbankkomponente.
14. OperatingSystem – Verwendet Nmap [Lyo14], um möglichst viele Informationen über das Betriebssystem zu ermitteln, und ändert den Typ der Komponente entsprechend. Es erstellt aus diesen Informationen zusätzlich eine Komponente, welche die virtuelle Maschine bzw. den Server repräsentiert, auf dem das Betriebssystem läuft.
15. OSLinux – Nutzt SSH, um weitere Informationen über das Betriebssystem zu ermitteln, als Eigenschaften bereitzustellen und gegebenenfalls den Typ der Komponente entsprechend anzupassen.
16. OSLinuxUbuntu – Extrahiert mithilfe von SSH die auf dem Ubuntu mit dem integrierten Paketmanager installierten Pakete und fügt die Paketnamen als Eigenschaft zu der ETG-Komponente hinzu.
17. PHPModule – Extrahiert mithilfe von SSH die PHP-Konfigurationsdatei und installierten PHP-Module, die als angehängte Datei bzw. Eigenschaft der Komponente im ETG repräsentiert werden.
18. Server – Crawler-Plugin das sowohl auf physikalischen als auch auf virtuellen Servern ausgeführt werden kann und ermittelt, wo und wie die VM, bzw. der Server, betrieben wird, also zum Beispiel ob es sich um eine VM bei Microsoft Azure oder Amazon EC2 handelt (vgl. Abschnitt 5.2.4).
19. ServerAzure – Crawler-Plugin, das unter bestimmten Bedingungen die SSH-Zugangsdaten des Servers setzt.
20. ServerEC2 – Analog zu ServerAzure für Amazon EC2.
21. ServerOpenStack – Analog zu ServerAzure für OpenStack.
22. Webserver – Crawler-Plugin, das mithilfe der HTTP-Header und Httprint [Net05] den Typ und die Version des Webservers ermittelt.
23. WebserverApacheSSH – Ermittelt die installierten Apache-Module und den absoluten Pfad, unter welchem die ausgelieferten Seiten gespeichert sind.

24. WebserverTomcatHTTPManager – Ermittelt Detailinformationen über den Tomcat und bestimmt alle darauf installierten Anwendungen.
25. WebserverTomcatJMX – Bestimmt die genaue Version des Tomcat und des darunterliegenden Betriebssystems über JMX.
26. WebserverTomcatSSH – Analysiert die Tomcat-Konfigurationsdateien, hängt diese an die ETG-Komponente an und extrahiert eine Auswahl wichtiger Konfigurationsparameter als Eigenschaften der Komponente.
27. WSDL_PortType – Erstellt durch die Analyse der WSDL-Datei ETG-Komponenten für alle Dienste, welche diesen PortType implementieren, und speichert die in der WSDL enthaltenen Informationen bezüglich Binding und Port als Eigenschaften der Relationen.
28. WSDL – Analysiert eine WSDL-Datei und extrahiert deren PortTypes.

7.4 AROMA-Migrationsassistent

Der „AROMA-Migrationsassistent“ folgt der in Kapitel 3 eingeführten AROMA-Methode. In dieser Arbeit wurde die AROMA-Methode mit dem OASIS-Standard TOSCA (vgl. Abschnitt 2.4.3) umgesetzt. Das vom Migrationsassistenten implementierte Konzept zur Umsetzung und Automatisierung der AROMA-Methode mit TOSCA wurde in Kapitel 6 vorgestellt. Für jede Migration wird ein „Migrationsprojekt“ angelegt, das aus verschiedenen „Migrationslösungen“ bestehen kann. Eine Migrationslösung stellt dabei einen Durchlauf durch den Migrationsassistenten dar. Dies erlaubt verschiedene Migrationsszenarien für ein Migrationsprojekt durchzuspielen, diese zu bewerten, zu vergleichen und anschließend das erfolgversprechendste anzuwenden. Jedes Migrationsprojekt wird auf Basis eines ETGs durchgeführt, wobei der Migrationsassistent keine Änderungen am ETG vornimmt, um unerwünschte Seiteneffekte zu verhindern. Jede Migrationslösung wiederum besteht aus verschiedenen „Revisionen“, die automatisch im Hintergrund angelegt werden, wenn eine bestehende Migrationslösung verändert wird. Dies ermöglicht dem Benutzer jederzeit zu vorhergehenden Ständen der Migration zurückkehren.

Im Folgenden beschreibt Abschnitt 7.4.1 die Architektur des AROMA-Migrationsassistenten. Die Schritte, die der Migrationsassistent standardmäßig basierend auf dem Migrations-Framework implementiert, werden in Abschnitt 7.4.2 ausgeführt. Zum Abschluss wird in Abschnitt 7.4.3 gesondert auf die Realisierung der Variante 2 („Migration geschäftsprozessbasierter Anwendungen“) der AROMA-Methode eingegangen.

7.4.1 Architektur des Migrationsassistenten

Abbildung 7.7 zeigt die Architektur des Migrationsassistenten, deren Kern das erweiterbare „Migrations-Framework“ und die Implementierung der Schritte der AROMA-Methode sind. Jeder Schritt besteht aus zwei Architekturkomponenten, eine für die Benutzeroberfläche im Migrationsassistenten und eine für die entsprechende Geschäftslogik. Das Migrations-Framework stellt die Integrationslogik der Implementierung der verschiedenen Schritte der AROMA-Methode bereit, das heißt eine einheitliche Datenhaltung, Navigation zwischen den Schritten und eine gemeinsame Benutzeroberfläche. Schritte bekommen vom Migrations-Framework, in Abhängigkeit von Migrationsprojekt, Migrationslösung und Revision, ein Objekt übergeben, von welchem der aktuelle Zustand der Migration gelesen und mit dem dieser verändert

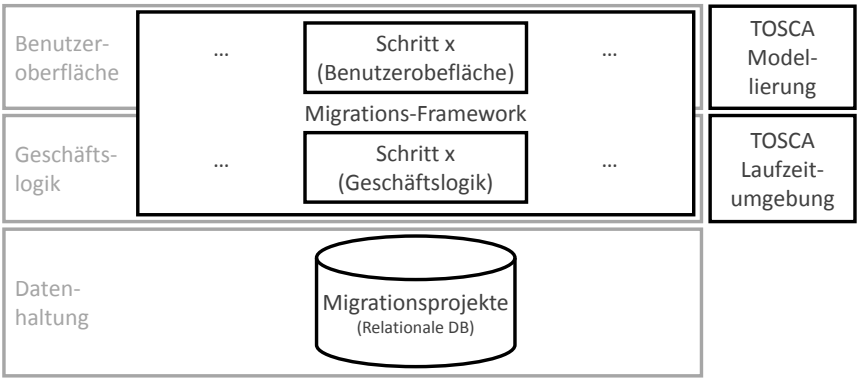

Abbildung 7.7: Architektur des Migrationsassistenten

werden kann. Die Kommunikation der Benutzeroberfläche intern und mit der Geschäftslogik ist nachrichtenbasiert implementiert. Dies ermöglicht es, in den vielen am Migrationsassistenten beteiligten Architekturkomponenten einfacher auf Änderungen und Nutzereingaben, zum Beispiel das Beginnen und Beenden eines Schrittes, zu reagieren. Auf der rechten Seite von Abbildung 7.7 befinden sich die Architekturkomponenten für die Modellierung und Ausführung von TOSCA, wobei die Modellierungskomponente, welche zur Darstellung von Anwendungstopologien und auch ETGs verwendet wird, nahtlos in die Benutzeroberfläche integriert ist. Dazu wurde das grafische TOSCA-Modellierungswerkzeug Winery [KBBL13] integriert und erweitert, damit es neben der Visualisierung von TOSCA-Anwendungstopologien auch für ETGs genutzt werden kann.

Die vorgestellte Architektur erlaubt die zukünftige Erweiterung des AROMA-Migrationsassistenten durch weitere Schritte, die im Migrations-Framework registriert werden. Die Reihenfolge der Schritte wird vom Framework dynamisch dadurch bestimmt, dass jeder Schritt angibt, nach welchen anderen Schritten er ausgeführt werden kann.

7.4.2 Realisierung des Migrationsassistenten

Dieser Abschnitt beschreibt die prototypische Implementierung der Schritte des AROMA-Migrationsassistenten. Der Migrationsassistent kann durch den Benutzer auf jedem beliebigen existierenden ETG im ETG-Framework gestartet werden, das heißt, der erste Schritt der AROMA-Methode wurde zu diesem Zeitpunkt schon mit dem ETG-Crawler aus Abschnitt 7.3 ausgeführt.

Anschließend wird mit Schritt 2 („Identifikation, Partitionierung und Isolation der Anwendung aus der Enterprise Topologie“) der AROMA-Methode fortgefahren, der als Benutzereingabe die Auswahl der zu migrierenden Komponenten entgegennimmt. Zur Auswahl der zu migrierenden Komponenten wird der ETG grafisch dargestellt und der Benutzer kann durch einen Klick auf die jeweilige Komponente diese zur Migration auswählen. Aus Gründen der Übersichtlichkeit kann zwischen der Ansicht des kompletten ETGs oder einer Ansicht, welche nur die Komponenten mit Anwendungsfunktionalität enthält, gewählt werden. Die Realisierung der Isolation baut auf dem in

Abschnitt 4.6 eingeführten Konzept auf. Dessen manuelle Schritte wurde nicht im Prototypen implementiert, da dies keine grundsätzlichen, neuen Probleme adressiert. Die Transformation nach TOSCA, Schritt 3 der AROMA-Methode, wird ohne Benutzereingaben im Hintergrund durchgeführt.

Für die Durchführung von Schritt 4 („Adaption an die konkrete Zielumgebung“) der AROMA-Methode bestimmt der Benutzer die Zielumgebung durch die Auswahl eines oder mehrerer Anbieter und deren Dienste. Anbieter und Dienste werden in einer XML-Datei beschrieben und können so einfach erweitert werden. Anschließend wird die Verteilung der zur Migration gewählten Komponenten auf die Zielumgebungen festgelegt. Um dies zu vereinfachen, generiert der Migrationsassistent dafür einen Vorschlag, der angezeigt wird und vom Benutzer angepasst (oder komplett verändert) werden kann. Nachdem die Verteilung feststeht, berechnet der Migrationsassistent mögliche Lösungen für die Adaption der Anwendungstopologie an die Zielumgebung und bewertet diese. Die besten Lösungen werden dem Benutzer zur Auswahl angeboten, wobei dieser zwischen den beiden in Abschnitt 6.3.2.3 eingeführten Bewertungsstrategien wählen kann, wie im Screenshot in Abbildung 7.8 dargestellt. Die vom Benutzer gewählte Lösung wird umgesetzt (vgl. Abschnitt 6.3.2.2) und für die Durchführung der Evaluation und manuellen Bearbeitung (Schritt 5 der AROMA-Methode) in das grafische TOSCA-Modellierungswerkzeug Winery importiert.

Im letzten Schritt hat der Benutzer die Möglichkeit, die paketierte CSAR (bzw. eine CSAR pro Zielumgebung) herunterzuladen (Schritt 6 der AROMA-Methode) und mithilfe einer entsprechenden TOSCA-Laufzeitumgebung bereitzustellen (Schritt 7 der AROMA-Methode). Die Erstellung der CSAR wird durch Winery implementiert, in welche die Anwendungstopologie im vorhergehenden Schritt importiert wurde.

Anschließend ist es möglich, eine weitere Anwendung aus dem gleichen ETG zu migrieren. Die Variante 1 („Selektive Erstellung der Enterprise Topologie“) der AROMA-Methode wurde, wie in Abschnitt 5.1.5.4 beschrieben, implementiert. Beim Start eines Crawler-Laufes kann der Benutzer in der Benutzeroberfläche eine der Crawling-Strategien, welche verschiedene Realisierungen der Funktion *selektionSicherstellen* darstellen, auswählen.

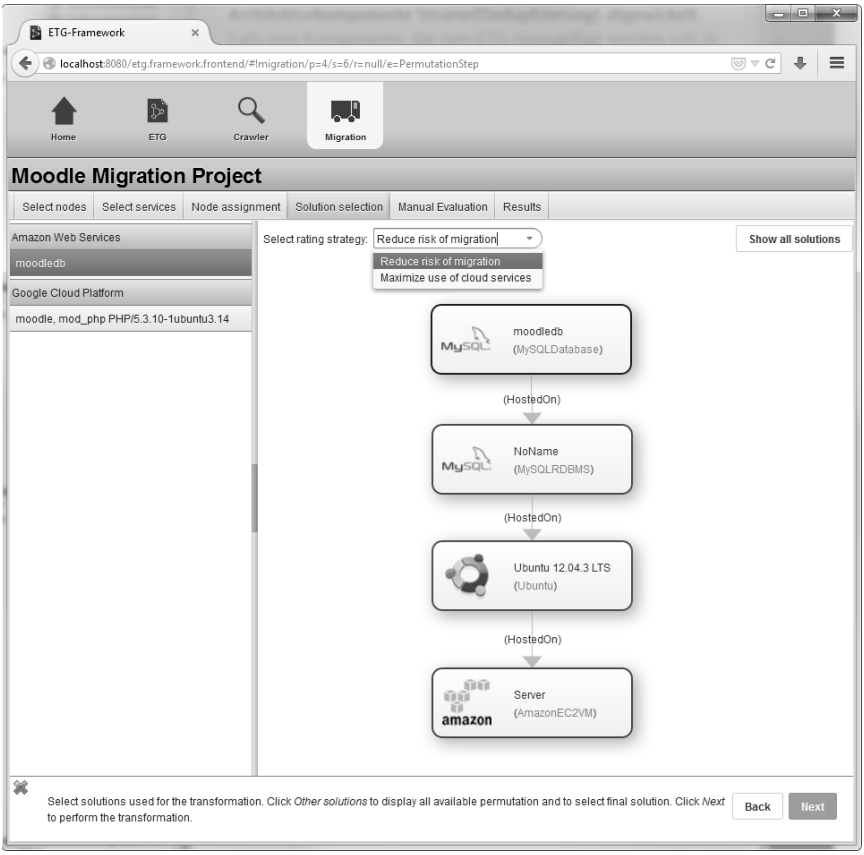

Abbildung 7.8: Visualisierung einer Lösung und Auswahl verschiedener Bewertungsstrategien im AROMA-Migrationsassistenten

Aktuell sind zwei Crawling-Strategien implementiert: (i) Das selektive Crawling einer gegebenen Anwendung (vgl. Variante 1 der AROMA-Methode) und (ii) das unbeschränkte Crawling aller Komponenten, für die Crawler-Plugins vorliegen (normaler Ablauf der AROMA-Methode). Weitere Crawling-Strategien können durch den Crawler-Administrator hinzugefügt werden. Die Erweiterungen zur Realisierung der Variante 2 („Migration geschäftsprozessbasierter Anwendungen“) wird in Abschnitt 7.4.3 beschrieben.

7.4.3 Realisierung von Variante 2 der AROMA-Methode

Variante 2 („Migration geschäftsprozessbasierter Anwendungen“) der AROMA-Methode sieht eine domänenspezifische Visualisierung und Selektion der zu migrierenden Komponenten basierend auf einem Geschäftsprozess vor. Der AROMA-Migrationsassistent implementiert Variante 2 für BPEL [OAS07]. Diese Variante kann beim Start des Migrationsassistenten gewählt werden, sobald sich mindestens ein Geschäftsprozess im zu migrierenden ETG befindet. Dazu wird der Geschäftsprozess, wie in Abbildung 7.9 dargestellt, visualisiert. Zur Umsetzung der Visualisierung von BPEL-Geschäftsprozessen wurde „ViPro“ [NKL+12] integriert und ein entsprechendes Visualisierungs-Template erstellt. Der Benutzer kann pro Aktivität wählen, ob – und falls ja in welche Zielumgebung – die Realisierung der Aktivität migriert werden soll. Diese Auswahl wird dann entsprechend propagiert und die Anwendungstopologie aufgeteilt, falls sich keine Widersprüche ergeben. In dieser Variante erlaubt es der Migrationsassistent die Identifikation und Isolation der zu migrierenden Komponenten anhand des Geschäftsprozesses durchzuführen. Die Adaption des Geschäftsprozesses selbst kann nach Abschluss des AROMA-Migrationsassistenten mit den Arbeiten von Kahlaf, Kopp, Wagner und Leymann realisiert werden [KL06, KKL08, WKL11].

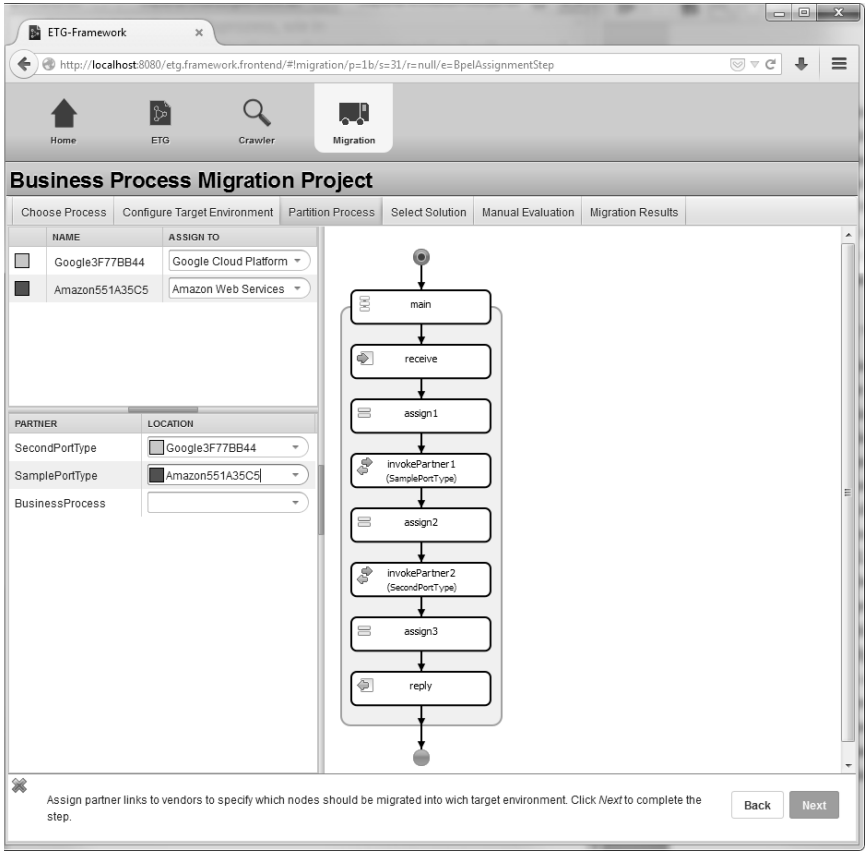

Abbildung 7.9: Visualisierung des Geschäftsprozesses mit ViPro (rechts) und Verteilung der zu migrierenden Aktivitäten auf Zielumgebungen (links)

KAPITEL 8

VALIDIERUNG UND EVALUATION

Die Validierung der AROMA-Methode diskutiert anhand verschiedener Fallstudien die Reproduzierbarkeit und Einsetzbarkeit auf verschiedensten Enterprise Topologien und die Migration daraus ausgewählter Anwendungen. Die Fallstudie „Moodle“ in Abschnitt 8.1 validiert dabei die gesamte Migration, die beiden Fallstudien in Abschnitt 8.2 beziehen sich auf die Validierung des ETGs und des ETG-Crawlers in anderen Arbeiten.

Die abschließende qualitative Evaluation in Abschnitt 8.3 diskutiert anhand der Kriterien (i) Automatisierung, (ii) Korrektheit, (iii) Verbesserung der Nutzung der Cloud-Eigenschaften der Anwendung, (iv) Verbesserung der Portabilität der Anwendung, (v) Anwendbarkeit und (vi) Erweiterbarkeit die Konzepte und Prototypen der vorliegenden Arbeit.

8.1 Fallstudie: Migration der Anwendung „Moodle“

Die Schulverwaltungs- und Lernanwendung „Moodle“ [Moo14] ist der zentrale Anwendungsfall des vom BMWi-geförderten Projektes „CloudCycle“, welches die Nutzung von Cloud Computing unter Beachtung von Sicherheits- und Compliance-Anforderungen in deutschen Schulen und anderen Einrichtungen der öffentlichen Hand ermöglichen will. Die Ausgangslage dieser Fallstudie ist, dass sich eine Schule entschieden hat, Moodle aus ihrer lokalen, virtualisierten IT in eine Cloud zu migrieren. Mit der Migration wurde ein Dienstleister beauftragt, der die AROMA-Methode anwendet. Der Benutzer des AROMA-Migrationsassistenten ist somit ein Mitarbeiter des Dienstleisters.

In Schritt 1 („Crawling der IT“) der AROMA-Methode wird der ETG der IT gecrawlt. Dazu erstellt der Benutzer in der ETG-Verwaltung ein neues ETG-Projekt und darin eine Komponente vom Typ *Application*. Diese repräsentiert Moodle und besitzt als einzige Eigenschaft die URL, unter der die Moodle-Benutzeroberfläche im Webbrowser erreichbar ist. Anschließend wird der ETG-Crawler gestartet und der den in Abbildung 8.1 gezeigten ETG automatisiert erstellt. Neben den hervorgehobenen Komponenten und Relationen von Moodle enthält der ETG eine Vielzahl weiterer Komponenten, von denen in Abbildung 8.1 zwei exemplarisch dargestellt sind.

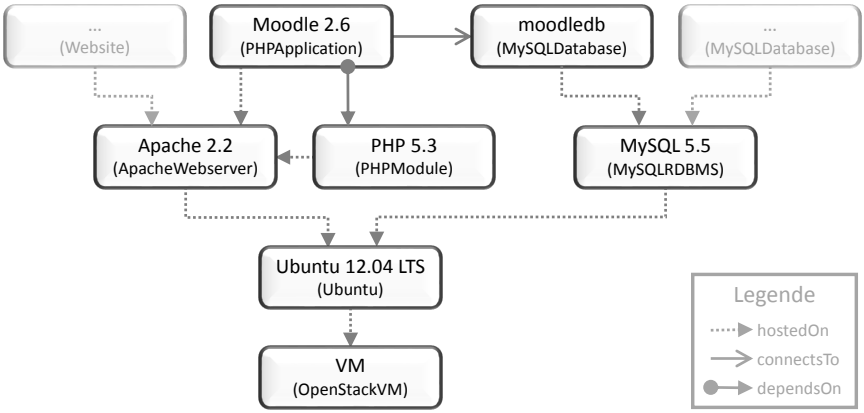

Abbildung 8.1: Enterprise Topologie Graph der Moodle enthält

Auf diesem Enterprise Topologie Graph startet der Benutzer den AROMA-Migrationsassistenten und wählt die Komponenten, welche er migrieren möchte. Im Rahmen dieser Fallstudie wird die Moodle-Komponente gewählt, worauf im Hintergrund alle für deren Betrieb notwendigen Komponenten im ETG identifiziert und isoliert werden (Schritt 2 der AROMA-Methode). Dies entspricht den in Abbildung 8.1 hervorgehobenen Komponenten und Relationen. Anschließend werden diese Komponenten im Hintergrund nach TOSCA transformiert (Schritt 3 der AROMA-Methode).

Als Eingabe für Schritt 4 („Adaption an die konkrete Zielumgebung“) der AROMA-Methode konfiguriert der Benutzer die gewünschte Zielumgebung, indem er entsprechende Anbieter und Dienste wählt. In dieser Fallstudie sind das die Anbieter Amazon und Google, wobei es auch möglich wäre, einzelne Dienste der Anbieter zu selektieren. Anschließend ordnet der Benutzer im AROMA-Migrationsassistenten die Komponenten mit Anwendungsfunktionalität einem Anbieter zu. Hier wurde entschieden die Anwendung zu partitionieren indem die Datenbank zukünftig bei Google und die anderen Teile der Anwendung bei Amazon betrieben werden (vgl. Abbildung 8.2). Auf Basis dieser Eingaben erstellt der AROMA-Migrationsassistent pro Anbieter verschiedene Lösungen für die Migration und bewertet diese anhand der in Abschnitt 6.3.2.3 eingeführten Bewertungsstrategien. Abbildung 8.3 zeigt zwei Lösungen für die Adaption der Komponenten welche zu Amazon migriert werden sollen. Die linke Seite zeigt eine Lösung mit möglichst wenig Risiko (Bewertungsstrategie i), in der die Anwendung nicht grundlegend verändert wurde. Bei dieser Lösung wird Moodle weiterhin auf einer virtuellen Maschine betrieben, da so die Konfiguration der Komponenten aus der Ursprungsumgebung direkt übernommen werden kann. Auf der rechten Seite zeigt Abbildung 8.3 eine Lösung für den gleichen Teil der Anwendung, welche jedoch die Eigenschaften der Zielumgebung möglichst stark nutzt (Bewertungsstrategie ii in Abschnitt 6.3.2.3). In diesem Fall wird die Anwendung auf dem Plattformdienst *Amazon Beanstalk* betrieben, der beispielsweise Elastizität und nutzungsabhängige Bezahlung ermöglicht. Der Benutzer entscheidet sich bei den Teilen von Moodle, die in der Zielumgebung *Amazon* betriebenen werden sollen, für die Lösung mit dem geringsten Risiko. Im Gegensatz dazu wählt er für die Datenbank die maxi-

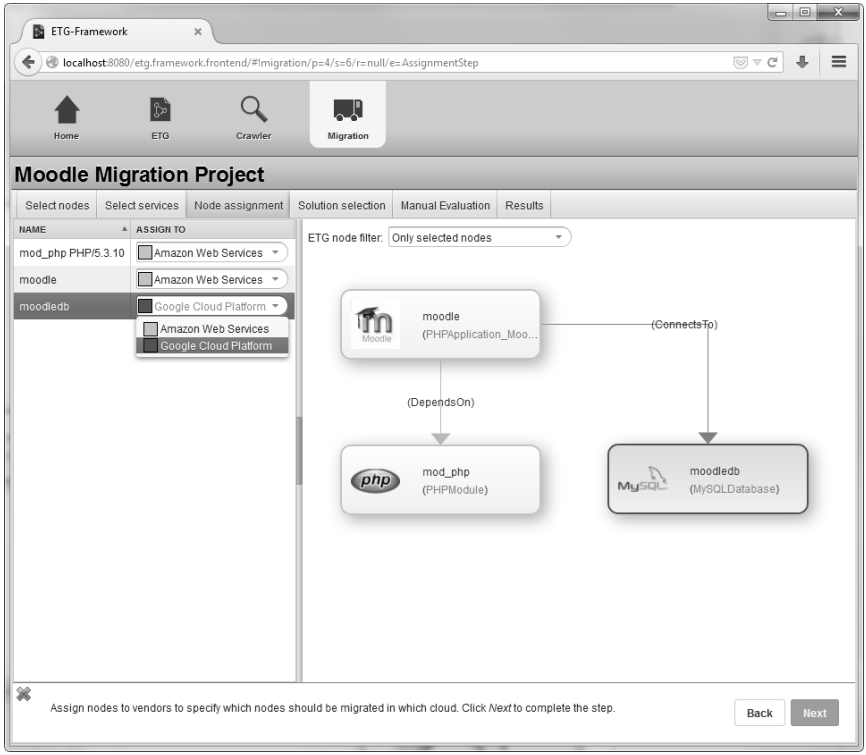

Abbildung 8.2: Zuweisung der gewählten Anbieter zu den Komponenten

male Ausnutzung der Zielumgebung, weshalb diese auf dem Plattformdienst „Google Cloud SQL“ betrieben werden soll. Falls dem Benutzer keine der Lösungen zusagt, kann er jederzeit in einen der vorhergehenden Schritte zurückkehren und zum Beispiel einen anderen Anbieter oder eine andere Verteilung wählen. Damit ist Schritt 4 der AROMA-Methode abgeschlossen.

Zur manuellen Evaluation (Schritt 5 der AROMA-Methode) nutzt der Benutzer das integrierte TOSCA-Modellierungswerkzeug Winery, mit dem er die Anwendungstopologie für die Zielumgebung prüfen und verändern kann. In dieser Fallstudie wählt der Benutzer eine andere Hardwarekonfiguration für die virtuelle Maschine, ergänzt die entsprechenden Zugangsdaten und hinterlegt in Winery den entsprechenden Plan für die Bereitstellung. Nach

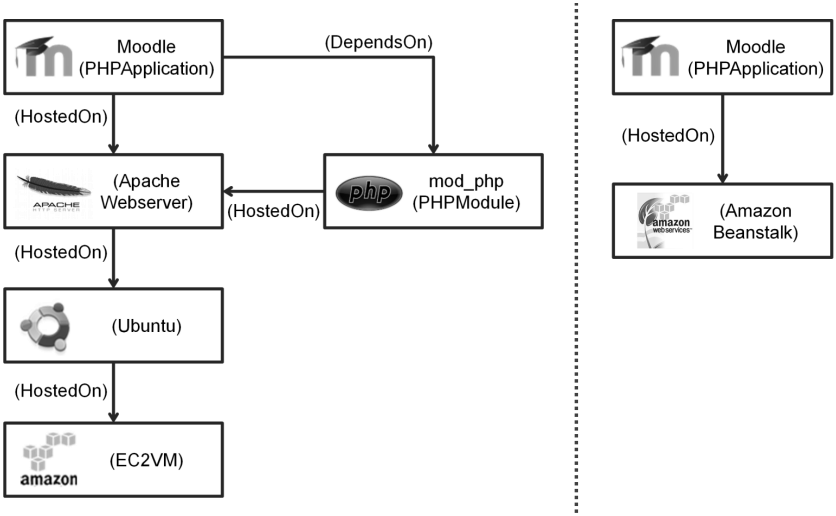

Abbildung 8.3: Verschiedene Lösungstopologien

Abschluss der Evaluation werden dem Benutzer die resultierenden CSARs angeboten (Schritt 6 der AROMA-Methode), jeweils eine pro Zielumgebung (d.h. Anbieter) oder eine anbieterübergreifende. Die heruntergeladene CSAR kann mithilfe von OpenTOSCA in der Zielumgebung bereitgestellt werden (Schritt 7 der AROMA-Methode). Die nachfolgenden Teilschritte (Test, Datenmigration, Umstellung und Außerbetriebnahme) wurden nicht im Rahmen der vorliegenden Arbeit umgesetzt (vgl. Abschnitt 3.2.7).

Diese Fallstudie zeigt, wie die Anwendung Moodle aus einer virtualisierten Umgebung in die Cloud migriert wird. Der Benutzer muss dafür nur die in diesem Abschnitt beschriebenen Eingaben tätigen, alles Weitere wird durch den AROMA-Migrationsassistenten automatisiert durchgeführt. Dabei werden verschiedene Zielumgebungen und Kombinationen von Diensten bewertet und evaluiert. Durch die Wahl eines Plattformdienstes für die Datenbank konnte erreicht werden, dass die Verwaltung, insbesondere Datensicherung und das Einspielen von Software-Aktualisierungen, durch einen externen Anbieter durchgeführt wird. Durch die Bereitstellung als CSAR wird zudem die Portabilität der Anwendung ermöglicht.

8.2 Verwendung von Enterprise Topologie Graphen und deren Crawling in anderen Arbeiten

Dieser Abschnitt stellt exemplarisch zwei der in Abschnitt 4.7 aufgelisteten Arbeiten vor, welche den Enterprise Topologie Graphen (Beitrag 2) und dessen Crawling (Beitrag 3) als Grundlage für die Analyse, Adaption und Optimierung der IT verwenden.

8.2.1 Deklarative Verwaltung und Adaption laufender Cloud-Anwendungen

Breitenbücher et al. [BBKL13b, BBL14] präsentieren einen Ansatz zur deklarativen Verwaltung und Adaption laufender Cloud-Anwendungen. Dabei wählt der Benutzer ein „Automatisiertes Management Pattern“ aus, um das Instanzmodell der Anwendung in Form eines ETGs in ein deklaratives Managementmodell zu überführen. Dieses wird anschließend in einen Management Workflow transformiert und ausgeführt. Wie in Abbildung 8.4 gezeigt, baut dieser Ansatz direkt auf dem ETG als Instanzmodell auf. Der ETG-Crawler, links in Abbildung 8.4, erstellt dabei den ETG der Anwendung.

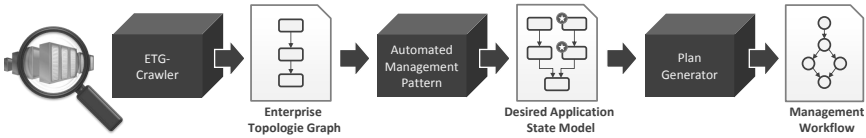

Abbildung 8.4: Übersicht des Ansatzes von Breitenbücher et al. (modifiziert aus [BBL14])

8.2.2 Reengineering von Geschäftsprozessen anhand ökologischer Faktoren

Nowak et al. [NBLU13] beschreiben ein Konzept zur Analyse der ökologischen Auswirkungen von Geschäftsprozessen, in diesem Fall deren Stromverbrauch. Dazu wird der Stromverbrauch der Komponenten ermittelt, die für die Ausführung des Geschäftsprozesses nötig sind. Anschließend wird dieser

auf die einzelnen Aktivitäten, das heißt Dienstaufrufe, des Geschäftsprozesses abgebildet. Die Propagierung des Stromverbrauchs, ausgehend von den Infrastrukturkomponenten, die den Strom konsumieren, wird auf einem ETG durchgeführt. Dazu ist es nötig, für einen gegebenen Geschäftsprozess alle Komponenten zu identifizieren, die für den Betrieb dieses Geschäftsprozesses nötig sind. Dies wird umgesetzt, indem der ETG-Crawler mit diesem Geschäftsprozess als Einstiegskomponente ausgeführt wird. Durch die Analyse des Geschäftsprozesses mithilfe der Arbeiten von Nowak et al. kann beispielsweise bestimmt werden, ob es sinnvoll ist Teile des Geschäftsprozesses und dessen Realisierung in eine Umgebung zu migrieren, die geringere ökologische Auswirkungen hat. Die Entscheidung, einen Teil des Geschäftsprozesses auszulagern, kann wiederum eine Eingabe von Schritt 2 der Variante 2 der AROMA-Methode „Migration geschäftsprozessbasierter Anwendungen“ darstellen, welche die Anwendung dementsprechend aufteilt und migriert.

Nowak et al. [NBL14] zeigen darüber hinaus, wie die automatische Kompensation des verursachten CO_2-Ausstoßes eines Geschäftsprozesses mit den in Abschnitt 8.2.1 vorgestellten Konzepten von Breitenbücher et al. kombiniert werden kann, welche auch auf dem ETG basieren.

8.3 Evaluation und Schlussfolgerungen

Dieser Abschnitt evaluiert die Konzepte und Prototypen anhand der Aspekte: (i) Automatisierung, (ii) Korrektheit, (iii) Verbesserung der Nutzung der Cloud-Eigenschaften der Anwendung, (iv) Verbesserung der Portabilität der Anwendung, (v) Anwendbarkeit und (vi) Erweiterbarkeit.

8.3.1 Automatisierung

Die Automatisierung aller Schritte unter Einbeziehung des Benutzers für Rückfragen und der manuellen Überprüfung kritischer, automatisiert erstellter Zwischenergebnisse wurde durchgehend umgesetzt. Durch die Automatisierung zuvor manueller Aufgaben sowie die Möglichkeit, Wissen

von Experten, zum Beispiel für einen Komponententyp, wiederverwendbar festzuhalten und automatisch einzusetzen, können die Migrationskosten deutlich gesenkt werden [SWH10, Hug13]. Des Weiteren reduziert die Automatisierung das Risiko von menschlichen Fehlern in der manuellen Durchführung [OGP03]. Insbesondere die hohe Anzahl an zu migrierenden Anwendungen spricht für die Automatisierung der Anwendungsmigration [Sha12]. Daher war die Automatisierung ein Kriterium der Bewertung aller Methoden und Prototypen dieser Arbeit, insbesondere der AROMA-Methode (Abschnitt 3.4), Crawler-Methode (Abschnitt 5.5) und der Prototypen (Kapitel 7). Das ETG-Framework und der AROMA-Migrationsassistent bündeln den gesamten Prozess und alle Beiträge der vorliegenden Arbeit unter einer einheitlichen Oberfläche. Diese kontrolliert die Benutzerinteraktion, das heißt die Benutzereingaben und das manuelle Prüfen von Zwischenergebnissen, und führt im Hintergrund alle automatisierbaren Aufgaben der Migration ohne Eingreifen des Benutzers durch.

Auch die Bereitstellung der Anwendung selbst wird durch deren Umsetzung als TOSCA-Anwendungstopologie und die entsprechende TOSCA-Laufzeitumgebung automatisiert. Durch die Verwendung von TOSCA wird darüber hinaus auch die Basis dafür geschaffen, um zukünftige Verwaltungsaufgaben der migrierten Anwendung automatisieren zu können.

8.3.2 Korrektheit

Die Anwendungsmigration ist korrekt, wenn die Anwendung in der Zielumgebung lauffähig ist und deren Funktionalität erhalten bleibt. Die AROMA-Methode macht die Erhaltung der Funktionalität wahrscheinlich, da sie Komponenten mit Anwendungsfunktionalität nicht austauscht oder verändert. Ein weiterer zentraler Aspekt für die Korrektheit der Migration ist die Qualität des ETGs (vgl. Abschnitt 3.4), welche durch die in Abschnitt 5.4.6 diskutierten Maßnahmen gewährleistet wird. Da jedoch die funktionale Äquivalenz der Anwendung in Ursprungs- und Zielumgebung nicht allgemeingültig nachgewiesen werden kann, ist es nötig, die Anwendung nach der Bereitstellung zu testen, bevor diese produktiv verwendet wird.

8.3.3 Verbesserung der Nutzung der Cloud-Eigenschaften

Die in Abschnitt 2.3.1 definierten Cloud-Eigenschaften motivieren den Betrieb einer Anwendung in der Cloud, auch da sie zu einer deutlichen Kostenreduzierung führen. Dieser Abschnitt zeigt, dass durch eine Migration mit der AROMA-Methode die Ausnutzung der Cloud-Eigenschaften einer Anwendung verbessert wird, teilweise so weit, dass die Migration in einer nativen Cloud-Anwendung resultiert (vgl. Definition 2.3). Dazu wird im Folgenden betrachtet, wie die Nutzung der einzelnen Cloud-Eigenschaften von der AROMA-Methode beeinflusst wird.

Die Elastizität einer Anwendung sagt aus, wie diese auf sich ändernde Ressourcen reagiert (vgl. Abschnitt 2.3.1). Pahl und Xiong [PX13] beschreiben, dass insbesondere die Statuslosigkeit der (meisten) Komponenten und das Externalisieren von Daten für die Elastizität einer Anwendung wichtig sind, weil diese zum Beispiel ermöglichen, neue Instanzen einer Anwendung hinzuzufügen. Wenn eine Anwendung ihre Daten in einer MySQL-Datenbank verwaltet, kann diese beispielsweise im Verlauf der Migration von einer lokalen virtuellen Maschine auf einen skalierbaren Plattformdienst wie „Amazon RDS“ [Ama14b] migriert werden. Neben der Sicherstellung der Elastizität muss der Kunde sich dann auch nicht mehr um Verwaltungsaufgaben wie Datensicherung, Überwachung, Softwareupdates und Vieles mehr kümmern. Diese Adaption kann in der Anwendungstopologie durch den Austausch des NodeTypes umgesetzt werden (siehe Abschnitt 6.3.2). Falls die Implementierung der Anwendung statuslos ist, kann diese auf einer elastischen Laufzeitumgebung betrieben werden und eine native Cloud-Anwendung wäre erreicht. Im Gegensatz dazu ist eine Anwendung, die Daten in einer Datei auf der lokalen Festplatte verwaltet, deutlich schwerer adaptierbar, insbesondere automatisiert, weil hier der Code der Datenhaltung signifikant verändert werden muss. Die Verbesserung der Elastizität hängt somit von der in Abschnitt 3.4.1 diskutierten Eignung der Anwendung ab. Die Modifikation der Anwendung hin zu einer Architektur, welche die automatische Verbesserung der Elastizität erlaubt, ist nicht Teil dieser Arbeit.

Durch die Realisierung der AROMA-Methode mit TOSCA werden die beiden Cloud-Eigenschaften Automatisierung (siehe dazu auch Abschnitt 8.3.1) und Selbstbedienung verbessert. Da die resultierende Anwendung als CSAR verfügbar ist, dem TOSCA-Paketformat, kann diese von einer entsprechenden TOSCA-Laufzeitumgebung automatisch bereitgestellt werden [BBLS12, BBKL14a]. Auch die anschließende Bereitstellung der Anwendung in einem Selbstbedienungsportal ist möglich, beispielsweise in der Vinothek [BBKL14e], die Teil des OpenTOSCA-Ökosystems ist. Darüber hinaus lassen sich mit TOSCA auch der Bereitstellung nachfolgende Verwaltungsaufgaben und die Terminierung der Anwendung automatisieren.

Die nutzungsabhängige Bezahlung wird nicht von der AROMA-Methode beeinflusst, sondern durch die in der Anwendungstopologie verwendeten Komponenten der Zielumgebung bestimmt. Das in der vorliegenden Arbeit beschriebene Vorgehen erlaubt es jedoch, die von einer Anwendung genutzten Komponenten so zu wählen, dass eine nutzungsabhängige Bezahlung möglich ist. Beispielsweise erlaubt die Migration von Anwendungskomponenten auf einen entsprechenden Infrastruktur- oder Plattformdienst, wie bei der Elastizität diskutiert, die nutzungsabhängige Bezahlung.

Insgesamt bleibt festzuhalten, dass die Migration einer Anwendung mit der AROMA-Methode unterstützt, die Nutzung der Cloud-Eigenschaften der migrierten Anwendung zu verbessern. Davon können auch Anwendungen profitieren, die in eine andere Zielumgebung als die Cloud migriert werden.

8.3.4 Verbesserung der Portabilität

Durch die Migration mit den vorgestellten Methoden und Prototypen wird die zukünftige Portabilität der migrierten Anwendung nachhaltig verbessert. Aufgrund des Crawlings eines Instanzmodells durch den ETG-Crawler steht – für viele Anwendungen erstmals überhaupt – ein aktuelles, vollständiges und technisch detailliertes Instanzmodell bereit. Die nachfolgende Transformation nach TOSCA und somit die Nutzung eines modellbasierten Ansatzes für die Bereitstellung und Verwaltung von Anwendungen vereinfacht zukünftige Migrationen deutlich und reduziert das Risiko eines Vendor-lock-ins [SJ12].

Durch die Nutzung eines OASIS-Standards wie TOSCA wird zudem die zukünftige Nutzung anderer, TOSCA-basierter Werkzeuge ermöglicht und die Abhängigkeit von einzelnen Herstellern reduziert. Darüber hinaus wird die Interoperabilität mit anderen in TOSCA vorliegenden Anwendungstopologien unterstützt.

8.3.5 Anwendbarkeit

Die Migration aller Arten von Anwendungen erfordert, möglichst wenige Annahmen über die Komponenten, Relationen, Architektur und Funktionsweise einer Anwendung zu treffen. Die AROMA-Methode erfüllt dies, indem sie sich beispielsweise nicht auf eine Sprache zur Beschreibung von Anwendungstopologien festlegt – ausschließlich in der Realisierung wird TOSCA verwendet. Enterprise Topologie Graphen, als zugrunde liegendes Metamodell für die Instanzmodelle, nutzen ein flexibles und wenig restriktives Typsystem und erlauben somit die Repräsentation von IT in verschiedensten Strukturen und Granularitäten. Der ETG-Crawler verwendet typspezifische Plugins zur Extraktion der benötigten Daten, so dass die Art und Weise der Extraktion nicht eingeschränkt ist. Die AROMA-Methode wird im Prototyp mit TOSCA realisiert, welches ein erweiterbares Typsystem beinhaltet und jede Art von Anwendung abbilden kann. Die Validierung in diesem Kapitel hat gezeigt, wie die verschiedenen Methoden und Konzepte auf Fallstudien angewendet wurden. Insbesondere der für die Realisierung der AROMA-Methode wichtige ETG und ETG-Crawler wurden in unterschiedlichen Anwendungsfällen validiert. Die Anforderungen der AROMA-Methode an mögliche Zielumgebungen für migrierte Anwendungen werden in Abschnitt 3.4.2 diskutiert.

Crawler-Plugins extrahieren die Informationen, welche für einen gegebenen Typ *Standard* sind, zum Beispiel die von einer Datenbank oder einem Webserver vorgesehenen Konfigurationsmöglichkeiten. Die anwendungsspezifischen Teile, zum Beispiel das Datenbankschema, muss das Crawler-Plugin nicht verstehen, es reicht dieses zu extrahieren und an die ETG-Komponente anzuhängen. Bei Komponenten mit Anwendungsfunktionalität, zum Beispiel in Java oder PHP implementierte Anwendungsfunktionalität, kann

auch die Implementierung für den ETG wichtige Informationen enthalten, beispielsweise aufgerufene Dienste oder Datenbankverbindungen. Für die Konfiguration dieser Komponenten und die Definition von Verbindungsinformationen gibt es Mechanismen in Sprach-Frameworks, zum Beispiel *Datasources* in JavaEE [Ora10], und Best Practices auf Industrie- oder Firmenebene, zum Beispiel die Verwendung der Datei `config.php` zur Konfiguration einer PHP-Anwendung. Diese Mechanismen sind damit wohlbekannt und können in typspezifischen Crawler-Plugins zur Extraktion dieser Informationen aus vielen unterschiedlichen Komponenten verwendet werden. Teilweise werden diese Best Practices jedoch nicht angewendet und für den ETG wichtige Informationen auf proprietäre, anwendungsspezifische Art und Weise kodiert. Um diese trotzdem zu extrahieren, sind drei unterschiedliche Ansätze vorgesehen und im ETG-Crawler umsetzbar: (i) Die generische Analyse der Implementierung, um die gesuchten Informationen basierend auf Heuristiken zu finden. Joukov et al. [JTO+11] zeigen einen Ansatz für JavaEE-basierte Anwendungen, die im Bytecode nach Abhängigkeiten auf Ressourcen suchen, zum Beispiel Datenbanken, Message Queues oder Dateien. Eines der im Rahmen der vorliegenden Arbeit entwickeltes Crawler-Plugin verfolgt einen ähnlichen Ansatz für PHP-Anwendungen, um die Konfiguration der Datenbankverbindung zu extrahieren (vgl. Abschnitt 7.3.3). Heuristiken haben jedoch zwangsläufig Grenzen [JTO+11], da immer Fälle auftreten werden, in denen diese Aspekte so kodiert wurden, dass die genutzte Heuristik fehlschlägt bzw. erweitert werden müsste. (ii) In diesem Fall gibt es den Ansatz, die Existenz der Relation, zum Beispiel einer Datenbankverbindung, zu bestimmen, beispielsweise durch die Analyse des Netzwerkverkehrs (siehe Abschnitt 2.6.3). Falls dann weitere Informationen benötigt werden, die nicht generisch gefunden werden können, werden diese beim Crawler-Administrator erfragt. (iii) Als dritten Ansatz sieht die Crawler-Methode auch die Bereitstellung von anwendungsspezifischen Crawler-Plugins vor. Für besonders wichtige Anwendungen oder solche mit komplexen Konfigurationsmöglichkeiten ist somit eine spezifische und flexible Verarbeitung möglich.

8.3.6 Erweiterbarkeit

Die im vorhergehenden Abschnitt diskutierte Anwendbarkeit der Methoden und Prototypen ist nur durch deren Erweiterbarkeit erreichbar. Je einfacher die Erweiterbarkeit, desto höher die Wahrscheinlichkeit, dass die entsprechenden Crawler-Plugins auch vorhanden sind. In einer *offenen Welt* dürfen und können keine Annahmen über aktuelle und zukünftige Arten von Komponenten, Relationen und Anwendungen getroffen werden. Somit kann im Voraus nie jede Art von Semantik bekannt sein. Dem wurde in der vorliegenden Arbeit durch den Einsatz von Erweiterungsmechanismen wie Plugins Rechnung getragen, wie diskutiert in der Architektur des ETG-Frameworks in Abschnitt 7.1, der ETG-Verwaltung in Abschnitt 7.2, des ETG-Crawlers in Abschnitt 7.3 und des AROMA-Migrationsassistenten in Abschnitt 7.4. Die Erweiterbarkeit der AROMA-Methode hinsichtlich der Unterstützung neuer Typen von Komponenten und Relationen wurde für Schritt 1 in Abschnitt 5.4.3, für Schritt 2 in Abschnitt 4.6.4, für Schritt 3 in Abschnitt 6.3.1, für Schritt 4 in Abschnitt 6.3.2.4 und für Schritt 7 in Abschnitt 6.3.5 diskutiert. Die Schritte 5 und 6 sind unabhängig von den in der zu migrierenden Anwendung verwendeten Typen.

Auf der Kehrseite verlangt dies, dass die entsprechenden Erweiterungen vorhanden sind, das heißt im Zweifelsfall vom Nutzer erstellt werden müssen. Die Herausforderung war also, das richtige Gleichgewicht zu finden und alle generisch lösbaren Fragestellungen allgemein und auf alle Ausprägungen von Anwendungen übertragbar zu beantworten. Dadurch ist es in manchen Fällen angebracht, eine Rückfrage an den Benutzer zu stellen, statt den Aufwand zur Entwicklung der zur Beantwortung der Fragen nötigen Crawler-Plugins zu investieren.

KAPITEL 9

ZUSAMMENFASSUNG UND AUSBLICK

Die automatisierte Migration von Anwendungen in eine Zielumgebung, beispielsweise die Cloud, ist ein wichtiger Aspekt der ständig fortschreitenden Adaption der IT an sich ändernde Gegebenheiten. Um diese durchzuführen ist ein Instanzmodell der gesamten IT erforderlich, also aller Komponenten und deren Relationen untereinander. Die vorliegende Arbeit ermöglicht einen technisch detaillierten Einblick in die IT einer Organisation (Forschungsschwerpunkt 1) und zeigt, wie dieser zur automatisierten Migration von Anwendungen genutzt werden kann (Forschungsschwerpunkt 2).

Abschnitt 9.1 betrachtet die beiden Forschungsschwerpunkte und ordnet die wissenschaftlichen Beiträge der Arbeit ein. Abschnitt 9.2 gibt davon ausgehend einen Ausblick auf mögliche weitergehende Arbeiten und Anwendungen der Forschungsergebnisse in anderen Bereichen.

9.1 Zusammenfassung der Forschungsbeiträge

Der erste Forschungsschwerpunkt befasst sich mit der Repräsentation, Verarbeitung und dem Crawling von Instanzmodellen, die einen technisch detaillierten, generischen, aktuellen und alle Arten von Anwendungsbestandteilen umfassenden Einblick in die gesamte IT geben. Zur Repräsentation, Verarbeitung und Analyse wurden Enterprise Topologie Graphen (ETGs) eingeführt (Beitrag 2), die auch schon in verschiedenen anderen Arbeiten Anwendung finden. Beitrag 2 umfasst auch die Formalisierung von ETGs und die Einführung wiederverwendbarer Operationen zu deren Verarbeitung. Beitrag 3 befasst sich mit dem automatisierten Crawling von IT-Instanzmodellen, die als ETGs repräsentiert werden. Die Crawler-Methode verfolgt dabei einen Plugin-basierten Ansatz, der es ermöglicht, alle aktuellen und zukünftigen Typen von Komponenten und Relationen zu unterstützen, und dabei auch die Integration bestehender Informationsquellen erlaubt. Das automatisierte Crawling von ETGs ist sowohl für eine erfolgreiche Migration als auch deren Nutzung in anderen Anwendungsfällen, zum Beispiel der Verwaltung laufender Anwendungen oder deren Optimierung hinsichtlich ökologischer Kriterien, entscheidend. Da ein solches Instanzmodell heutzu-

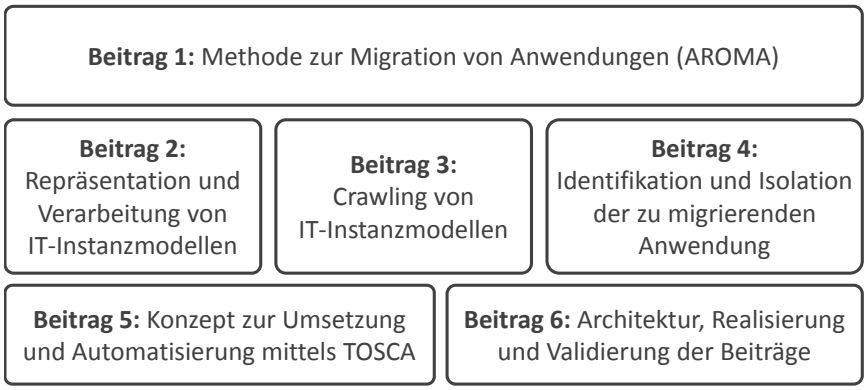

Abbildung 9.1: Übersicht der Forschungsbeiträge

tage meist nicht vorliegt, weder als ETG noch in einer anderen Form, werden komplexe Analyse- und Adaptionsaufgaben bisher oft auf Basis veralteter oder manuell recherchierter Informationen bewältigt. Diesem Aspekt wurde hier eine besondere Bedeutung beigemessen, da das Vorliegen eines solch weitreichenden Instanzmodells neue Wege und Möglichkeiten eröffnet, wie der zweite Forschungsschwerpunkt zeigt.

Der zweite Forschungsschwerpunkt umfasst die automatisierte Migration von Anwendungen in eine andere Umgebung, in dieser Arbeit veranschaulicht durch deren Migration in die Cloud. Beitrag 1 führt dazu die AROMA-Methode ein, welche die zu migrierende Anwendung in einem ETG identifiziert und extrahiert, in eine Anwendungstopologie transformiert, für die Zielumgebung adaptiert und nach einer manuellen Evaluation durch den Benutzer bereitstellt. Dies geschieht auf dem vom ersten Forschungsschwerpunkt gecrawlten ETG, der im Laufe der AROMA-Methode in eine Anwendungstopologie transformiert und weiterverarbeitet wird. Vom ETG ausgehend befasst sich Beitrag 4 mit der Identifikation der Komponenten, die für den Betrieb der zu migrierenden Anwendungsfunktionalität nötig sind. Dazu gehört auch die Isolation der Komponenten aus ihrer Umgebung, also die Behandlung von Relationen zu Komponenten, die nicht zwangsläufig für den Betrieb nötig sind. Die automatisierte Migration von Anwendungen ermöglicht es, von den Vorteilen fortschrittlicher IT-Umgebungen, wie beispielsweise der Cloud, zu profitieren, ohne diese Anwendungen komplett neu entwickeln oder manuell migrieren zu müssen. Neben der Automatisierung liegt der Fokus darauf, die Funktionalität der Anwendung zu erhalten.

Die vorliegende Arbeit macht sich für die Anpassung an die Zielumgebung, die Evaluation durch den Benutzer und die Bereitstellung der Anwendung Fortschritte bei den Sprachen zur Beschreibung von Anwendungstopologien zunutze: Neue Sprachen, beispielsweise TOSCA, ermöglichen die automatisierte Bereitstellung und Verwaltung sowie Portabilität und Interoperabilität der Anwendung. Beitrag 5 umfasst daher ein Konzept, wie die AROMA-Methode mit TOSCA als Sprache zur Beschreibung von Anwendungstopologien umgesetzt werden kann, und skizziert die dafür nötige Werkzeugunterstützung. Dazu gehört auch die Anpassung der Anwendung

an die Zielumgebung, wobei diese nur so weit adaptiert wird, dass sie in der Zielumgebung lauffähig ist. Im Zentrum stehen dabei die Erhaltung der Funktionalität und die Reduzierung des Migrationsrisikos. Dem gegenüber steht eine zweite, durch den Benutzer wählbare, Adaptionsstrategie, welche die Nutzung von Cloud-Diensten maximiert.

Das zentrale Rahmenwerk zur Integration aller auf dem ETG basierenden Funktionalitäten ist das ETG-Framework. Dessen erweiterbare und Service-orientierte Architektur ist Teil von Beitrag 6. Darauf aufbauend wurden die Architekturen der ETG-Verwaltung, des Migrationsassistenten, welcher die AROMA-Methode mit TOSCA realisiert, und des ETG-Crawlers, der die Crawler-Methode realisiert, entwickelt. Alle Beiträge und Prototypen wurden anhand verschiedener Fallstudien validiert und basierend auf ausgewählten Kriterien evaluiert. Die vorliegende Arbeit ermöglicht die *technische und automatisierte Durchführung* einer Anwendungsmigration, ein Forschungsbereich, der nach Jamshidi et al. [JAP13] im Gegensatz zu Vorgehensmodellen und Systemen zur Entscheidungsunterstützung bisher nicht ausreichend betrachtet wurde.

9.2 Ausblick

Dieser Abschnitt skizziert mögliche zukünftige Forschungsarbeiten und diskutiert andere Bereiche, in denen die vorgestellten Forschungsergebnisse eingesetzt werden können.

Die Adaption der Anwendung in Schritt 4 der AROMA-Methode macht die Anwendungstopologie in der Zielumgebung lauffähig. Weitergehende Optimierungen der Anwendung, beispielsweise hinsichtlich deren Architektur, Skalierbarkeit oder Mehrmandantenfähigkeit, sind nicht Teil der Migration, sondern deren Weiterentwicklung [AGW05, SWH10]. Von großem Interesse sind hier Ansätze, die Anwendungstopologien automatisiert adaptieren und optimieren, so dass diese die Vorteile der jeweiligen Zielumgebung stärker nutzen können. In Bezug auf die Cloud als Zielumgebung bedeutet dies, möglichst viele Cloud-Anwendungen in native Cloud-Anwendungen umzuwandeln. Die AROMA-Methode legt hierzu die Grundlage, da die zu

migrierende Anwendung am Ende als Anwendungstopologie vorliegt und zukünftige Arbeiten diese zur Optimierung und Adaption nutzen können.

Die Migration der Daten wurde in dieser Arbeit nicht betrachtet, ist aber ein wichtiger Aspekt jeder Migration. Ein möglicher Ansatz ist, die Anwendungsdaten auf Basis der Informationen im ETG zu identifizieren und zu extrahieren, so dass diese in die resultierende Anwendungstopologie eingefügt werden können. Nach der Bereitstellung können die Anwendungsdaten dann wieder importiert werden. Ein ähnlicher Ansatz ist die Generierung eines TOSCA-Plans, der die Datenmigration, also die Extraktion in der Ursprungsumgebung und den Import in der Zielumgebung, durchführt. Die Herausforderung dabei ist es, alle Daten korrekt zu identifizieren und generisch zu migrieren.

Um die Kosteneinsparungen der Migration in vollem Umfang realisieren zu können, ist es nötig, die Anwendung in der Ursprungsumgebung außer Betrieb zu nehmen. Hierbei ist die Herausforderung, zu identifizieren, welche Komponenten nicht mehr benötigt werden und wie diese automatisiert außer Betrieb genommen werden können. Dies könnte auf Basis des ETGs und unter Verwendung der Operation zur Isolation der zu migrierenden Anwendung umgesetzt werden.

Neben der Anwendungsmigration wurden die Beiträge dieser Arbeit bereits in verschiedenen anderen Forschungsbereichen eingesetzt. Weitere zukünftige Forschungsbereiche, in denen diese Beiträge angewandt werden könnten, werden im Folgenden kurz angerissen: Die Sicherstellung der Regelkonformität der IT, zu internen und externen Vorgaben, könnte auf dem ETG analysiert werden, beispielsweise der Ort, an dem Daten gespeichert und verarbeitet werden dürfen. Genauso könnte geprüft werden, ob komplexe Adaptionen der IT wie geplant durchgeführt wurden. Beim Outsourcing und bei Firmenübernahmen stellt sich die Frage, wie viele Anwendungen, Server oder Geschäftsprozesse in der jeweiligen Organisation vorhanden sind und wie diese untereinander zusammenhängen. Hier können der ETG und der ETG-Crawler helfen, eine Inventarisierung der IT durchzuführen, um auf dieser Basis die nötigen Entscheidungen zu treffen.

LITERATURVERZEICHNIS

[Abe06] Aberdeen Group. The Legacy Application Modernization Benchmark Report. Technischer Bericht, 2006. (Zitiert auf Seite 16)

[ABLS13] V. Andrikopoulos, T. Binz, F. Leymann, S. Strauch. How to adapt applications for the Cloud environment. Computing, Springer, 95(6):493–535, 2013. (Zitiert auf den Seiten 26, 36, 37 und 63)

[ADNM+12] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu, A. Gericke, C. Sheridan. MODAClouds: A model-driven approach for the design and execution of applications on multiple Clouds. In Proceedings of the 4th ICSE Workshop on Modeling in Software Engineering, S. 50–56. 2012. (Zitiert auf Seite 65)

[AFG+09] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. Above the Clouds: A Berkeley View of Cloud Computing. Technischer Bericht UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009. (Zitiert auf den Seiten 16, 33, 34 und 35)

[AFG+10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia. A View of Cloud Computing. Communications of the ACM, 53(4):50–58, 2010. (Zitiert auf Seite 34)

[AGW05] E. Ackermann, R. Gimnich, A. Winter. Ein Referenz-Prozess der Software-Migration. Softwaretechnik-Trends, 25(4):20–22, 2005. (Zitiert auf den Seiten 58, 61, 84 und 210)

[AKSSR11] S. Al-Kiswany, D. Subhraveti, P. Sarkar, M. Ripeanu. VMFlock: Virtual Machine Co-migration for the Cloud. In Proceedings of the 20th International Symposium on High Performance Distributed Computing, HPDC '11, S. 159–170. 2011. (Zitiert auf Seite 64)

[Ama14a] Amazon Web Services, Inc. Amazon CloudFormation, 2014. URL http://aws.amazon.com/cloudformation/. (Zitiert auf den Seiten 40 und 156)

[Ama14b] Amazon Web Services, Inc. Relational Database Service (Amazon RDS), 2014. URL http://aws.amazon.com/rds/. (Zitiert auf Seite 201)

[Ama14c] Amazon Web Services, Inc. Simple Storage Service (Amazon S3), 2014. URL http://aws.amazon.com/s3/. (Zitiert auf Seite 179)

[Apa04] Apache Software Foundation. Apache License, Version 2.0, 2004. URL http://www.apache.org/licenses/LICENSE-2.0. (Zitiert auf Seite 165)

[Apa11] Apache Software Foundation. Apache Synapse – Documentation, 2011. (Zitiert auf Seite 146)

[Apa14a] Apache Software Foundation. Apache Camel Documentation, 2014. (Zitiert auf den Seiten 146 und 148)

[Apa14b] Apache Software Foundation. Apache Derby – open source relational database, 2014. URL http://db.apache.org/derby/. (Zitiert auf Seite 179)

[Apa14c] Apache Software Foundation. Apache Tomcat 7 – Documentation, 2014. URL http://tomcat.apache.org/tomcat-7.0-doc. (Zitiert auf Seite 144)

[ASL13a] V. Andrikopoulos, Z. Song, F. Leymann. Supporting the Migration of Applications to the Cloud through a Decision Support System.

In Proceedings of the 6th IEEE International Conference on Cloud Computing, S. 565–572. 2013. (Zitiert auf Seite 76)

[ASL13b] V. Andrikopoulos, S. Strauch, F. Leymann. Decision Support for Application Migration to the Cloud: Challenges and Vision. In Proceedings of the 3rd International Conference on Cloud Computing and Service Science, S. 149–155. 2013. (Zitiert auf Seite 76)

[ASLW14] V. Andrikopoulos, S. G. Sáez, F. Leymann, J. Wettinger. Optimal Distribution of Applications in the Cloud. In M. Jarke, J. Mylopoulos, C. Quix, Herausgeber, Proceedings of the 26th Conference on Advanced Information Systems Engineering, S. 75–90. 2014. (Zitiert auf den Seiten 38 und 76)

[BBG+14] G. Breiter, M. Behrendt, M. Gupta, S. Moser, R. Schulze, I. Sippli, T. Spatzier. Software defined environments based on TOSCA in IBM cloud implementations. IBM Journal of Research and Development, 58(2):1–10, 2014. (Zitiert auf Seite 40)

[BBH+13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, S. Wagner. OpenTOSCA – A Runtime for TOSCA-based Cloud Applications. In Proceedings of the 11th International Conference on Service-Oriented Computing, S. 692–695. 2013. (Zitiert auf den Seiten 24, 27, 156 und 165)

[BBK+12] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, D. Schumm. Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA. In Proceedings of the 20th International Conference on Cooperative Information Systems, S. 416–424. 2012. (Zitiert auf den Seiten 27, 42, 90 und 165)

[BBK+13] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann, A. Weiß. Improve Resource-Sharing through Functionality-Preserving Merge of Cloud Application Topologies. In Proceedings of the 3rd International Conference on Cloud Computing and Service Science, S. 96–103. 2013. (Zitiert auf den Seiten 24 und 138)

[BBK+14a] U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann, J. Wettinger. Combining Declarative and Imperative Cloud Applicati-

on Provisioning based on TOSCA. In Proceedings of the IEEE International Conference on Cloud Engineering, S. 87–96. 2014. (Zitiert auf den Seiten 25, 165 und 173)

[BBK+14b] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, M. Wieland. Context-aware Cloud Application Management. In Proceedings of the 4th International Conference on Cloud Computing and Services Science, S. 499–509. 2014. (Zitiert auf den Seiten 26, 49, 67, 117, 134 und 172)

[BBKL13a] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann. Automated Discovery and Maintenance of Enterprise Topology Graphs. In Proceedings of the 6th IEEE International Conference on Service Oriented Computing & Applications, S. 126–134. 2013. (Zitiert auf Seite 24)

[BBKL13b] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann. Pattern-based Runtime Management of Composite Cloud Applications. In Proceedings of the 3rd International Conference on Cloud Computing and Service Science. 2013. (Zitiert auf den Seiten 26, 115 und 198)

[BBKL14a] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann. Advanced Web Services, Kapitel TOSCA: Portable Automated Deployment and Management of Cloud Applications, S. 527–549. Springer, 2014. (Zitiert auf den Seiten 24, 41, 43 und 202)

[BBKL14b] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann. Migration of enterprise applications to the cloud. it – Information Technology, Special Issue: Architecture of Web Application, 56(3):106–111, 2014. (Zitiert auf Seite 24)

[BBKL14c] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann. Automating Cloud Application Management Using Management Idioms. In Proceedings of the 6th International Conferences on Pervasive Patterns and Applications, S. 60–69. 2014. (Zitiert auf den Seiten 25, 76 und 134)

[BBKL14d] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann. The OpenTOSCA Ecosystem, 2014. URL `http://www.opentosca.org`. (Zitiert auf Seite 165)

[BBKL14e] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann. Vinothek - A Self-Service Portal for TOSCA. In N. Herzberg, M. Kunze, Herausgeber, Proceedings of the 6th Central-European Workshop on Services and their Composition, Band 1140 von CEUR Workshop Proceedings, S. 69–72. 2014. (Zitiert auf den Seiten 27, 165 und 202)

[BBL14] U. Breitenbücher, T. Binz, F. Leymann. A Method to Automate Cloud Application Management Patterns. In Proceedings of the 8th International Conference on Advanced Engineering Computing and Applications in Sciences, S. 140–145. 2014. (Zitiert auf den Seiten 115 und 198)

[BBLS12] T. Binz, G. Breiter, F. Leymann, T. Spatzier. Portable Cloud Services Using TOSCA. IEEE Internet Computing, 16(03):80–85, 2012. (Zitiert auf den Seiten 21, 25, 41, 44 und 202)

[BBN+09] T. Bowden, B. Bauer, J. Nerin, S. Feng, S. Seibold. The /proc filesystem. Kernel Dokumentation, 2009. URL `https://www.kernel.org/doc/Documentation/filesystems/proc.txt`. (Zitiert auf Seite 56)

[BCD08] S. Basu, F. Casati, F. Daniel. Toward Web Service Dependency Discovery for SOA Management, S. 422–429. Institute of Electrical and Electronics Engineers, 2008. (Zitiert auf Seite 52)

[BCX+12] P. Beserra, A. Camara, R. Ximenes, A. Albuquerque, N. Mendonca. Cloudstep: A step-by-step decision process to support legacy application migration to the cloud. In Proceedings of the 6th IEEE International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, S. 7–16. 2012. (Zitiert auf den Seiten 62 und 63)

[BDF04] R. Black, A. Donnelly, C. Fournet. Ethernet topology discovery without network assistance. In Proceedings of the 12th IEEE International Conference on Network Protocols, S. 328–339. 2004. (Zitiert auf Seite 53)

[BFL+12] T. Binz, C. Fehling, F. Leymann, A. Nowak, D. Schumm. Formalizing the Cloud through Enterprise Topology Graphs. In Proceedings of the

5th IEEE International Conference on Cloud Computing, S. 742–749. 2012. (Zitiert auf den Seiten 25, 91 und 115)

[BGL07] M. Bowker, B. Garrett, B. Laliberte. EMC Smarts Application Discovery Manager, 2007. (Zitiert auf Seite 53)

[BGM+12] M. Buschle, S. Grunow, F. Matthes, M. Ekstedt, M. Hauder, S. Roth. Automating Enterprise Architecture Documentation using an Enterprise Service Bus. In Proceedings of the 18th Americas Conference on Information Systems, Band 6, S. 4213–4226. 2012. (Zitiert auf den Seiten 47 und 50)

[BGRV12] L. Badger, T. Grance, P.-C. R., J. Voas. Cloud Computing Synopsis and Recommendations - Recommendations of the National Institute of Standards and Technology. NIST Special Publication 800-146, 2012. (Zitiert auf den Seiten 34 und 73)

[BHJ+13] D. Betts, A. Homer, A. Jezierski, M. Narumoto, H. Zhang. Moving Applications to the Cloud on Windows Azure. Patterns & practices. Microsoft Developer Guidance, 3 Auflage, 2013. (Zitiert auf Seite 60)

[BHS+12] M. Buschle, H. Holm, T. Sommestad, M. Ekstedt, K. Shahzad. A Tool for Automatic Enterprise Architecture Modeling. In IS Olympics: Information Systems in a Diverse World, Band 107, S. 1–15. Springer, 2012. (Zitiert auf Seite 50)

[BKK01] A. Brown, G. Kar, A. Keller. An active approach to characterizing dynamic dependencies for problem determination in a distributed environment. In Proceedings of the 9th IEEE/IFIP International Symposium on Integrated Network Management, S. 377–390. 2001. (Zitiert auf den Seiten 50, 52 und 118)

[BLNS12] T. Binz, F. Leymann, A. Nowak, D. Schumm. Improving the Manageability of Enterprise Topologies Through Segmentation, Graph Transformation, and Analysis Strategies. In Proceedings of the 16th IEEE International Enterprise Distributed Object Computing Conference, S. 61–70. 2012. (Zitiert auf den Seiten 25, 50, 51, 91 und 115)

[BLS11] T. Binz, F. Leymann, D. Schumm. CMotion: A Framework for Migration of Applications into and between Clouds. In Proceedings of the 9th IEEE International Conference on Service-Oriented Computing and Applications, S. 225–228. 2011. (Zitiert auf den Seiten 25 und 38)

[BLW+97] J. Bisbal, D. Lawless, B. Wu, J. Grimson, V. Wade, R. Richardson, D. O'Sullivan. An overview of legacy information system migration. In Proceedings of the 4th Asia Pacific Software Engineering Conference, S. 529–530. 1997. (Zitiert auf den Seiten 59 und 60)

[BMM12] E. Brandtzæg, S. Mosser, P. Mohagheghi. Towards CloudML, a Model-based Approach to Provision Resources in the Clouds. In Proceedings of the 8th European Conference on Modelling Foundations and Applications, S. 18–27. 2012. (Zitiert auf Seite 39)

[BMR+10] S. Buckl, F. Matthes, S. Roth, C. Schulz, C. Schweda. A Conceptual Framework for Enterprise Architecture Design. In E. Proper, M. Lankhorst, M. Schönherr, J. Barjis, S. Overbeek, Herausgeber, Trends in Enterprise Architecture Research, Band 70 von Lecture Notes in Business Information Processing, S. 44–56. Springer Berlin Heidelberg, 2010. (Zitiert auf Seite 47)

[BS95] M. L. Brodie, M. Stonebraker. Migrating Legacy Systems: Gateways, Interfaces &Amp; the Incremental Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995. (Zitiert auf den Seiten 59 und 60)

[BS06] C. Batini, M. Scannapieco. Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications). Springer, 2006. (Zitiert auf den Seiten 119 und 120)

[BSB11] G. Breiter, T. Spatzier, M. Behrendt. Cloud Computing-An Industry Perspective. it – Information Technology, 53(4):165–172, 2011. (Zitiert auf Seite 32)

[Bui14] BuiltWith Pty Ltd. Blog Usage – Statistics for websites using Blog technologies, 2014. URL `http://trends.builtwith.com/cms/blog`. (Zitiert auf Seite 156)

[BW07] C. Braun, R. Winter. Integration of IT Service Management into Enterprise Architecture. In Proceedings of the 22th ACM Symposium on Applied Computing, S. 1215–1219. 2007. (Zitiert auf Seite 46)

[BYV+09] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 25(6):599–616, 2009. (Zitiert auf den Seiten 33 und 86)

[Can14] Canonical Ltd. Juju – Automate your cloud infrastructure, 2014. URL https://juju.ubuntu.com/. (Zitiert auf Seite 40)

[CB12] M. Chauhan, M. Babar. Towards Process Support for Migrating Applications to Cloud Computing. In Proceedings of the International Conference on Cloud and Service Computing, S. 80–87. 2012. (Zitiert auf den Seiten 62 und 73)

[CCMW01] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C, 2001. (Zitiert auf Seite 30)

[Cha99] J. Charles. Middleware moves to the forefront. IEEE Computer, 32(5):17–19, 1999. (Zitiert auf Seite 32)

[Cha04] D. Chappell. Enterprise service bus. O'Reilly Media, Inc., 2004. (Zitiert auf den Seiten 30 und 145)

[Cha12] V. Chalapathi. TOGAF vs Archimate, 2012. URL http://chalapathi-bantupalli.blogspot.de/2012/05/togaf-vs-archimate.html. (Zitiert auf Seite 47)

[Che14a] Chef Software, Inc. Automation platform for the new IT, 2014. URL http://www.getchef.com/. (Zitiert auf den Seiten 40, 156 und 157)

[Che14b] Chef Software, Inc. Ohai, 2014. URL http://docs.opscode.com/ohai.html. (Zitiert auf Seite 56)

[Cis12] Cisco. Cisco Global Cloud Networking Survey - Summary and Analysis of Results Worldwide Results, 2012. URL http://www.cisco.com/go/cloudsurvey. (Zitiert auf Seite 16)

[CKG+08] A. Caracas, A. Kind, D. Gantenbein, S. Fussenegger, D. Dechouniotis. Mining semantic relations using NetFlow. In Proceedings of the 3rd IEEE/IFIP International Workshop Business-driven IT Management, S. 110–111. 2008. (Zitiert auf Seite 53)

[Cla04] B. Claise. Cisco Systems NetFlow Services Export Version 9. Internet Engineering Task Force, RFC 3954, 2004. URL `http://tools.ietf.org/html/rfc3954`. (Zitiert auf Seite 53)

[Col11] Coleman Parkes Research. HP Cloud & Transformation Study, 2011. (Zitiert auf Seite 16)

[CTG+12] K. Cheng, S. Turkarslan, N. Garcia, S. Howard, S. Tinline-Jones, S. Pelluru, S. Coriani, J. A. Bravo. Migrating Data-Centric Applications to Windows Azure. Technischer Bericht, Microsoft, 2012. (Zitiert auf Seite 61)

[CZMB08] X. Chen, M. Zhang, Z. M. Mao, P. Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In Proceedings of the 8th USENIX conference on Operating systems design and implementation, Band 8, S. 117–130. 2008. (Zitiert auf Seite 53)

[Dan13] T. Dang. Open Services for Lifecycle Collaboration Reconciliation Specification Version 2.0, 2013. OSLC. (Zitiert auf den Seiten 138 und 139)

[DFG+11] I. Diaz, G. Fernandez, P. Gonzalez, M. Martin, J. Tourino. Extending the Globus Information Service with the Common Information Model. In Proceedings of the 9th IEEE International Symposium on Parallel and Distributed Processing with Applications, S. 113–119. 2011. (Zitiert auf Seite 48)

[DG08] J. Dean, S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, 51(1):107–113, 2008. (Zitiert auf Seite 33)

[DGST11] S. Dustdar, Y. Guo, B. Satzger, H.-L. Truong. Principles of Elastic Processes. IEEE Internet Computing, 15(5):66–71, 2011. (Zitiert auf Seite 34)

[DHM05] X. Dong, A. Halevy, J. Madhavan. Reference reconciliation in complex information spaces. In Proceedings of the 24th ACM International Conference on Management of data, S. 85–96. 2005. (Zitiert auf Seite 138)

[Die05] R. Diestel. Graph Theory. Graduate Texts in Mathematics. Springer, 2005. (Zitiert auf den Seiten 92 und 94)

[Dij76] E. Dijkstra. A discipline of programming. Prentice-Hall series in automatic computation. Prentice-Hall, Incorporated, 1976. (Zitiert auf Seite 68)

[Dis13] Distributed Management Taskforce. Common Information Model (CIM), 2.39.0 Auflage, 2013. URL `http://www.dmtf.org/standards/cim`. (Zitiert auf Seite 48)

[DMT13] A. Demange, N. Moha, G. Tremblay. Detection of SOA Patterns. In S. Basu, C. Pautasso, L. Zhang, X. Fu, Herausgeber, Service-Oriented Computing, Band 8274 von Lecture Notes in Computer Science, S. 114–130. Springer Berlin Heidelberg, 2013. (Zitiert auf Seite 85)

[Dom04] P. Domingos. Multi-relational record linkage. In Proceedings of the 3rd Workshop on Multi-Relational Data Mining, S. 31–48. 2004. (Zitiert auf Seite 138)

[DP09] S. Ducasse, D. Pollet. Software Architecture Reconstruction: A Process-Oriented Taxonomy. IEEE Transactions on Software Engineering, 35(4):573–591, 2009. (Zitiert auf Seite 51)

[DS13] L. DeMichiel, B. Shannon. JSR 342: Java Platform, Enterprise Edition 7 (Java EE 7) Specification, 2013. URL `https://jcp.org/en/jsr/detail?id=342`. (Zitiert auf Seite 176)

[EK02] C. Ensel, A. Keller. An Approach for Managing Service Dependencies with XML and the Resource Description Framework. Journal of Network and Systems Management, 10(2):147–170, 2002. (Zitiert auf Seite 53)

[Emm00] W. Emmerich. Software Engineering and Middleware: A Roadmap. In Proceedings of the 22nd International Conference on on Software Engineering, S. 117–129. 2000. (Zitiert auf Seite 32)

[Ens99] C. Ensel. Automated generation of dependency models for service management. In Workshop of the OpenView University Association. 1999. (Zitiert auf den Seiten 49, 50, 52 und 118)

[FAB+10] M. Farwick, B. Agreiter, R. Breu, M. Häring, K. Voges, I. Hanschke. Towards Living Landscape Models: Automated Integration of Infrastructure Cloud in Enterprise Architecture Management, S. 35–42. Institute of Electrical and Electronics Engineers, 2010. (Zitiert auf den Seiten 50 und 118)

[FAB+11a] M. Farwick, B. Agreiter, R. Breu, S. Ryll, K. Voges, I. Hanschke. Automation Processes for Enterprise Architecture Management. In Proceedings of the 15th International Enterprise Distributed Object Computing Conference Workshops, S. 340–349. 2011. (Zitiert auf Seite 51)

[FAB+11b] M. Farwick, B. Agreiter, R. Breu, S. Ryll, K. Voges, I. Hanschke. Requirements for Automated Enterprise Architecture Model Maintenance. In Proceedings of the 13th International Conference on Enterprise Information Systems. 2011. (Zitiert auf den Seiten 119, 120 und 121)

[FAW07] R. Fischer, S. Aier, R. Winter. A Federated Approach to Enterprise Architecture Model Maintenance. Enterprise Modelling and Information Systems Architectures, 2(2):14–22, 2007. (Zitiert auf Seite 51)

[FFH12] F. Fittkau, S. Frey, W. Hasselbring. CDOSim: Simulating cloud deployment options for software migration support. In Proceedings of the 6th IEEE International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, S. 37–46. 2012. (Zitiert auf Seite 75)

[FFH13] S. Frey, F. Fittkau, W. Hasselbring. Search-based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud. In Proceedings of the 35th International Conference on Software Engineering, S. 512–521. 2013. (Zitiert auf den Seiten 73 und 76)

[FGM+97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Internet Engineering Task Force, RFC 7231, 1997. URL `http://tools.ietf.org/html/rfc2068`. (Zitiert auf Seite 143)

[FH11a] S. Frey, W. Hasselbring. The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications. International Journal on Advances in Software, 4(3):342–353, 2011. (Zitiert auf den Seiten 36, 37, 62 und 65)

[FH11b] S. Frey, 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, S. 269–278. 2011. (Zitiert auf den Seiten 65 und 75)

[FHK+09] U. Frank, D. Heise, H. Kattenstroth, D. Ferguson, E. Hadar, M. Waschke. ITML: a domain-specific modeling language for supporting business driven it management. In Proceedings of the 9th Workshop on Domain-Specific Modeling at the International Conference on Object Oriented Programming, Systems, Languages and Applications. 2009. (Zitiert auf Seite 47)

[FLR+13] C. Fehling, F. Leymann, S. T. Ruehl, M. Rudek, 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, S. 9–16. 2013. (Zitiert auf den Seiten 59, 60, 63, 79, 85 und 86)

[FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter. Cloud Computing Patterns. Springer, 2014. (Zitiert auf den Seiten 34, 35, 36, 63 und 76)

[FLRS12] C. Fehling, F. Leymann, J. Rütschlin, D. Schumm. Pattern-Based Development and Management of Cloud Applications. Future Internet Special Issue "Recent Advances in Web Services", 4(1):110–141, 2012. (Zitiert auf Seite 67)

[Fow02] M. Fowler. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. (Zitiert auf Seite 31)

[FWS10] P. Fremantle, S. Weerawarana, L. de Silva. Cloud Native. online, 2010. URL http://pzf.fremantle.org/2010/05/cloud-native.html. (Zitiert auf den Seiten 36 und 37)

[FZRL08] I. Foster, Y. Zhao, I. Raicu, S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In Proceedings of the Grid Computing Environments Workshop, S. 1–10. 2008. (Zitiert auf Seite 87)

[Gar00] D. Garlan. Software architecture: a roadmap. In Proceedings of the 22nd International Conference on on Software Engineering,, ICSE '00, S. 91–101. 2000. (Zitiert auf Seite 51)

[GD02] D. Gantenbein, L. Deri. Categorizing computing assets according to communication patterns. Advanced Lectures on Networking, S. 491–495, 2002. (Zitiert auf Seite 54)

[GGJGT+12] S. Garcia-Gomez, M. Jimenez-Ganan, Y. Taher, C. Momm, F. Junker, J. Biro, A. Menychtas, V. Andrikopoulos, S. Strauch. Challenges for the comprehensive management of Cloud Services in a PaaS framework. Scalable Computing: Practice and Experience, 13(3), 2012. (Zitiert auf Seite 36)

[GGRTRC13] J. Garcıa-Galán, O. Rana, P. Trinidad, A. Ruiz-Cortés. Migrating to the Cloud: a Software Product Line based analysis. In Proceedings of the 3rd International Conference on Cloud Computing and Service Science, S. 149–155. 2013. (Zitiert auf Seite 75)

[Gmb13] B. I. GmbH. Duden – Anwendung, 2013. URL http://www.duden.de/rechtschreibung/Anwendung. (Zitiert auf Seite 31)

[GMR10] J. Garbani, T. Mendel, E. Radcliffe. The Writing on IT's Complexity Wall. Technischer Bericht, Forrester Research, 2010. (Zitiert auf Seite 15)

[Goo14] Google Inc. The JRE Class White List, 2014. URL https://developers.google.com/appengine/docs/java/jrewhitelist. (Zitiert auf Seite 65)

[Gra86] J. Gray. Why do computers stop and what can be done about it? In Proceedings of the 20th Symposium on reliability in distributed software and database systems, S. 3–12. 1986. (Zitiert auf den Seiten 38 und 69)

[Gru13] A. Grund. Complete Enterprise Topologies with routing information of Enterprise Services Buses to enable Cloud-migration. Diplomarbeit, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, 2013. (Zitiert auf Seite 149)

[GSH+07] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, B. Gao. A Framework for Native Multi-Tenancy Application Development and Management. In Proceedings of the 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, S. 551–558. 2007. (Zitiert auf Seite 36)

[GSK13] A. Gunka, S. Seycek, H. Kühn. Moving an Application to the Cloud: An Evolutionary Approach. In Proceedings of the 1st International Workshop on Multi-cloud Applications and Federated Clouds, S. 35–42. 2013. (Zitiert auf Seite 64)

[GVB11] S. Garg, S. Versteeg, R. Buyya. SMICloud: A Framework for Comparing and Ranking Cloud Services. In Proceedings of the 4th IEEE International Conference on Utility and Cloud Computing, S. 210–218. 2011. (Zitiert auf Seite 76)

[GWT14] GWT Open Source Project. Google Web Toolkit, 2014. URL `http://www.gwtproject.org/`. (Zitiert auf Seite 176)

[Ham14] J. Hamilton. The Cloud: Fastest Industry Transition Ever, 2014. URL `http://perspectives.mvdirona.com/2014/01/04/TheCloudFastestIndustryTransitionEver.aspx`. (Zitiert auf Seite 16)

[Hau05] A. Haug. Flexibles Neben- und Miteinander – Modernisierung mit dem smartShift-Verfahren. IT Fokus, 2005. (Zitiert auf Seite 16)

[HB04] H. Haas, A. Brown. Web Services Glossary - W3C Working Group Note. Technischer Bericht, World Wide Web Consortium - W3C, 2004. URL http://www.w3.org/TR/ws-gloss/. (Zitiert auf Seite 30)

[HBBL14] P. Hirmer, U. Breitenbücher, T. Binz, F. Leymann. Automatic Topology Completion of TOSCA-based Cloud Applications. In Proceedings of the Informatik 2014, Band P-232 von LNI, S. 247–258. 2014. (Zitiert auf den Seiten 26 und 172)

[HM11] J. Humble, J. Molesky. Why Enterprises Must Adopt Devops to Enable Continuous Delivery. Cutter IT Journal, 24(8), 2011. (Zitiert auf Seite 40)

[HMR12] M. Hauder, F. Matthes, S. Roth. Challenges for Automated Enterprise Architecture Documentation. In Proceedings of the 7th Workshop on Trends in Enterprise Architecture Research and Practice-Driven Research on Enterprise Transformation, S. 21–39. 2012. (Zitiert auf den Seiten 51, 120 und 121)

[HP12] HP. HP Universal Discovery software. Datenblatt, 2012. URL http://hp.com/go/UD. (Zitiert auf Seite 54)

[HSS+10] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, M. Tawarmalani. Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. ACM Special Interest Group on Data Communication, 40(4):243–254, 2010. (Zitiert auf den Seiten 66 und 73)

[Hug13] N. Hughes. Connecting the Dots in the Enterprise Technology Malaise, 2013. URL http://www.wired.com/insights/2013/06/connecting-the-dots-in-the-enterprise-technology-malaise/. (Zitiert auf den Seiten 16 und 200)

[Hum06] J. Humphreys. System Virtualization: Sun Microsystems Enables Choice, Flexibility, and Management. Technischer Bericht, IDC for Sun Microsystems, 2006. (Zitiert auf Seite 33)

[HW03] G. Hohpe, B. Woolf. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional, 2003. (Zitiert auf den Seiten 145, 146 und 147)

[IBM14] IBM Corp. Tivoli Application Dependency Discovery Manager, 2014. URL http://ibm.co/K9IDni. (Zitiert auf Seite 54)

[IEE90] IEEE. 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology, 1990. (Zitiert auf den Seiten 79, 151 und 153)

[IJCK06] S. Iyengar, A. Jadhav, S. Chakrabarti, V. Kedia. Data Integration, Entity Reconciliation and Search in Personal Information Networks. Technischer Bericht, ACM SIGKDD 2006, 2006. (Zitiert auf den Seiten 138 und 140)

[JAP13] P. Jamshidi, A. Ahmad, C. Pahl. Cloud Migration Research: A Systematic Review. IEEE Transactions on Cloud Computing, 1(2):142–157, 2013. (Zitiert auf den Seiten 36, 58, 59, 66, 67, 68, 69 und 210)

[JDBN+10] H. von Jouanne-Diedrich, J. Blechinger, C. P. Neumann, S. Schwarz, R. Lenz. Integration verteilter und heterogener Configuration-Management-Datenbanken. Informatik-Spektrum, 33(4):351–362, 2010. (Zitiert auf Seite 48)

[JDMV08] N. Joukov, M. V. Devarakonda, K. Magoutis, N. Vogl. Built-to-Order Service Engineering for Enterprise IT Discovery, S. 91–98. Institute of Electrical and Electronics Engineers, 2008. (Zitiert auf Seite 55)

[Joh13] Johan den Haan. The cloud landscape described, categorized, and compared, 2013. URL http://www.theenterprisearchitect.eu/blog/2013/10/12/the-cloud-landscape-described-categorized-and-compared/. (Zitiert auf Seite 33)

[Jos09] A. Josey. TOGAF Version 9.1 Enterprise Edition – An Introduction. The Open Group, 11, 2009. (Zitiert auf Seite 47)

[JPRD10] N. Joukov, B. Pfitzmann, H. V. Ramasamy, M. V. Devarakonda. Application-storage discovery. In Proceedings of the 3rd Annual Haifa Experimental Systems Conference, S. 1–14. 2010. (Zitiert auf den Seiten 56 und 121)

[JTO+11] N. Joukov, V. Tarasov, J. Ossher, B. Pfitzmann, S. Chicherin, M. Pistoia, T. Tateishi. Static discovery and remediation of code-embedded

resource dependencies. In Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management, S. 233–240. 2011. (Zitiert auf den Seiten 56 und 204)

[JVZS+14] A. Juan-Verdejo, S. Zschaler, B. Surajbali, H. Baars, H.-G. Kemper. InCLOUDer: A Formalised Decision Support Modelling Approach to Migrate Applications to Cloud Environments. In 40th Euromicro Conference on Software Engineering and Advanced Applications. 2014. (Zitiert auf Seite 76)

[KBBL12] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann. BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications. In Proceedings of the 4th Business Process Model and Notation Workshop, S. 38–52. 2012. (Zitiert auf den Seiten 27 und 173)

[KBBL13] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann. Winery – A Modeling Tool for TOSCA-based Cloud Applications. In Proceedings of the 11th International Conference on Service-Oriented Computing, S. 700–704. 2013. (Zitiert auf den Seiten 27, 165, 173 und 188)

[KBK00] A. Keller, U. Blumenthal, G. Kar. Classification and computation of dependencies for distributed management. In Proceedings of the 5th IEEE Symposium on Computers and Communications, S. 78–83. 2000. (Zitiert auf den Seiten 49 und 117)

[KBS04] D. Krafzig, K. Banke, D. Slama. Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. (Zitiert auf Seite 30)

[KGE06] A. Kind, D. Gantenbein, H. Etoh. Relationship Discovery with NetFlow to Enable Business-Driven IT Management. In Proceedings of the 1st IEEE/IFIP International Workshop Business-Driven IT Management, S. 63–70. 2006. (Zitiert auf Seite 53)

[KHSS10] A. Khajeh-Hosseini, I. Sommerville, I. Sriram. Research Challenges for Enterprise Cloud Computing. Technischer Bericht, LSCITS, 2010. (Zitiert auf Seite 73)

[KK01] A. Keller, G. Kar. Determining service dependencies in distributed systems. In Proceedings of the IEEE International Conference on Communications, Band 7, S. 2084–2088. 2001. (Zitiert auf Seite 53)

[KKL08] O. Kopp, R. Khalaf, F. Leymann. Deriving Explicit Data Links in WS-BPEL Processes. In Proceedings of the International Conference on Services Computing, S. 367–376. 2008. (Zitiert auf den Seiten 83 und 191)

[KL06] R. Khalaf, F. Leymann. Role-based Decomposition of Business Processes using BPEL. In Proceedings of the International Conference on Web Services, S. 770–780. 2006. (Zitiert auf den Seiten 83 und 191)

[Kru04] P. Kruchten. The Rational Unified Process: An Introduction. The Addison-Wesley object technology series. Addison-Wesley, 2004. (Zitiert auf Seite 61)

[KW03] F. Keller, 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, S. 173–182. 2003. (Zitiert auf Seite 38)

[LaC07] M. LaChance. Dirty Little Secrets of Application Dependency Mapping. online, 2007. URL http://www.itsmwatch.com/itil/article.php/3718406/Dirty-Little-Secrets-of-Application-Dependency-Mapping.htm. (Zitiert auf den Seiten 53, 55, 57 und 118)

[Ley09] F. Leymann. Cloud Computing: The Next Revolution in IT. In Proceedings of the 52th Photogrammetric Week. 2009. (Zitiert auf den Seiten 16, 31, 32, 33, 34, 35 und 87)

[Ley10] F. Leymann. BPEL vs. BPMN 2.0: Should You Care? In Proceedings of the 2nd International Workshop on BPMN, Lecture Notes in Business Information Processing, S. 8–13. 2010. (Zitiert auf Seite 31)

[LFM+11] F. Leymann, C. Fehling, R. Mietzner, A. Nowak, S. Dustdar. Moving Applications to the Cloud: An Approach based on Application Model Enrichment. International Journal of Cooperative Information Systems, 20(3):307–356, 2011. (Zitiert auf den Seiten 66 und 73)

[LL10] J. Ludewig, H. Lichter. Software Engineering: Grundlagen, Menschen, Prozesse, Techniken. Dpunkt.Verlag GmbH, 2010. (Zitiert auf den Seiten 46 und 151)

[LLM+08] L. Liu, Y. Li, Q. Ma, K. W. Sun, Y. Chen, H. Wang. Automatic model-based service hosting environment migration. In Proceedings of the IEEE Network Operations and Management Symposium, S. 682–685. 2008. (Zitiert auf Seite 64)

[LMPR07] D. Lindquist, H. Madduri, C. J. Paul, B. Rajaraman. IBM Service Management architecture. IBM Systems Journal, 46(3):423–440, 2007. (Zitiert auf Seite 48)

[LR00] F. Leymann, D. Roller. Production Workflow – Concepts and Techniques. Prentice Hall PTR, 2000. (Zitiert auf den Seiten 30 und 96)

[Lyo14] G. Lyon. Nmap - Free Security Scanner For Network Exploration & Security Audits, 2014. URL `http://nmap.org/`. (Zitiert auf den Seiten 56, 149 und 185)

[Mat10] R. Matteson. DepMap: Dependency Mapping of Applications Using Operating System Events. Diplomarbeit, California Polytechnic State University, San Luis Obispo, 2010. (Zitiert auf Seite 55)

[Mat13] M. Mathews. Are You Ready for Software-Defined Everything? Wired, 2013. URL `http://www.wired.com/2013/05/are-you-ready-for-software-defined-everything/`. (Zitiert auf Seite 40)

[MBS+10] P. Mohagheghi, A. J. Berre, A. Sadovykh, F. Barbier, G. Benguria. Reuse and migration of legacy systems to interoperable cloud services-the REMICS project. Proceedings of Mda4ServiceCloud, 10, 2010. (Zitiert auf Seite 65)

[McK09] N. McKeown. Software-defined Networking. Infocom Keynote Talk, 2009. (Zitiert auf Seite 40)

[MDJV08] K. Magoutis, M. Devarakonda, N. Joukov, N. G. Vogl. Galapagos: Model-driven discovery of end-to-end application-storage relationships in distributed systems. IBM Journal of Research and Development, 52(4+5):367–377, 2008. (Zitiert auf Seite 56)

[MDW+99] V. Machiraju, M. Dekhil, K. Wurster, P. Garg, M. Griss, J. Holland. Towards generic application auto-discovery. In Proceedings of the IEEE/IFIP Network Operations and Management Symposium, S. 75–87. 1999. (Zitiert auf den Seiten 47, 50, 55 und 118)

[MFKL10] R. Mietzner, C. Fehling, D. Karastoyanova, F. Leymann. Combining horizontal and vertical composition of services. In Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications, S. 1–8. 2010. (Zitiert auf Seite 31)

[MG09] P. Mell, T. Grance. The NIST Definition of Cloud Computing. Information Technology Laboratory, 2009. (Zitiert auf den Seiten 16, 31, 33, 34 und 35)

[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, Germany, 2010. (Zitiert auf Seite 39)

[MK01] N. C. Mendonça, J. Kramer. An Approach for Recovering Distributed System Architectures. Automated Software Engineering, 8(3–4):311–354, 2001. (Zitiert auf Seite 51)

[MKLT13] M. Menzel, M. Klems, H. A. Lê, S. Tai. A Configuration Crawler for Virtual Appliances in Compute Clouds. In Proceedings of the 6th IEEE International Conference on Cloud Engineering, S. 201–209. 2013. (Zitiert auf Seite 54)

[ML10] R. Mietzner, F. Leymann. A self-service portal for service-based applications. In Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications, S. 1–8. 2010. (Zitiert auf den Seiten 39 und 85)

[Moh11] T. S. Mohan. Migrating into a Cloud, Kapitel 2, S. 43–56. John Wiley & Sons, Inc., 2011. (Zitiert auf Seite 62)

[Moo14] Moodle.org. Moodle – Open-source learning platform, 2014. URL `https://moodle.org/`. (Zitiert auf den Seiten 90 und 194)

[MR12] M. Menzel, R. Ranjan. CloudGenius: Decision Support for Web Server Cloud Migration. In Proceedings of the 21st International Conference on World Wide Web, S. 979–988. 2012. (Zitiert auf Seite 76)

[MUTL09] R. Mietzner, T. Unger, R. Titze, F. Leymann. Combining Different Multi-tenancy Patterns in Service-Oriented Applications. In Proceedings of the 13th IEEE International Enterprise Distributed Object Computing Conference, S. 131–140. 2009. (Zitiert auf Seite 36)

[MZY+13] F. J. Meng, X. Zhuo, B. Yang, J. M. Xu, P. Jin, A. Apte, J. Wigglesworth. A Generic Framework for Application Configuration Discovery with Pluggable Knowledge. In Proceedings of the 6th IEEE International Conference on Cloud Computing, S. 236–243. 2013. (Zitiert auf Seite 55)

[NBF+12] A. Nowak, T. Binz, C. Fehling, O. Kopp, F. Leymann, S. Wagner. Pattern-driven Green Adaptation of Process-based Applications and their Runtime Infrastructure. Computing, S. 463–487, 2012. (Zitiert auf Seite 26)

[NBL14] A. Nowak, U. Breitenbücher, F. Leymann. Automating Green Patterns to Compensate CO2 Emissions of Cloud-based Business Processes. In Proceedings of the 8th International Conference on Advanced Engineering Computing and Applications in Sciences, S. 132–139. 2014. (Zitiert auf Seite 199)

[NBLU13] A. Nowak, T. Binz, F. Leymann, N. Urbach. Determining Power Consumption of Business Processes and their Activities to Enable Green Business Process Reengineering. In Proceedings of the 17th IEEE International Enterprise Distributed Object Computing Conference, S. 259–266. 2013. (Zitiert auf den Seiten 26, 115 und 198)

[Neo14] Neo Technology, Inc. Neo4j – Graphs for Everyone, 2014. URL `http://neo4j.com/`. (Zitiert auf Seite 179)

[Net05] Net-square Solutions Pvt Ltd. httprint web server fingerprinting tool, 2005. URL `http://www.net-square.com/httprint.html`. (Zitiert auf den Seiten 56, 144 und 185)

[Nie11] D. Nielsen. Bringing on-demand, self-service, scalable, and measurable together in the clouds, 2011. URL http://www.ibm.com/developerworks/podcast/dwi/feature040611-dnielsen.html. (Zitiert auf den Seiten 34 und 35)

[NKAJ59] H. Newcombe, J. Kennedy, S. Axford, A. James. Automatic linkage of vital records. Science, 130(3381):954–959, 1959. (Zitiert auf den Seiten 137, 138 und 139)

[NKL+12] A. Nowak, D. Karastoyanova, F. Leymann, A. Rapoport, D. Schumm. Flexible Information Design for Business Process Visualizations. In Proceedings of the 5th IEEE International Conference on Service Oriented Computing and Applications, S. 1–8. 2012. (Zitiert auf Seite 191)

[OAS07] OASIS. Web Services Business Process Execution Language Version 2.0. OASIS, 2007. URL https://www.oasis-open.org/committees/wsbpel/. (Zitiert auf den Seiten 30 und 191)

[OAS13a] OASIS. Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0. OASIS, 2013. URL http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.html. (Zitiert auf den Seiten 43 und 164)

[OAS13b] OASIS. Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. OASIS, 2013. URL http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html. (Zitiert auf den Seiten 21, 39, 41, 92 und 165)

[OD08] D. L. Olson, D. Delen. Advanced Data Mining Techniques. Springer, 2008. (Zitiert auf den Seiten 119 und 120)

[OGP03] D. Oppenheimer, A. Ganapathi, 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. 2003. (Zitiert auf den Seiten 45, 67 und 200)

[OMG11a] OMG. Business Process Model and Notation (BPMN) Version 2.0, 2011. URL http://www.omg.org/spec/BPMN/2.0/. (Zitiert auf Seite 30)

[OMG11b] OMG. Unified Modeling Language (UML), 2011. URL http://www.omg.org/spec/UML. (Zitiert auf den Seiten 39 und 47)

[OMG13] OMG. Architecture-Driven Modernization, 2013. URL http://adm.omg.org/. (Zitiert auf Seite 65)

[Ora10] Oracle. Resource Connections – The Java EE 5 Tutorial, 2010. URL http://docs.oracle.com/javaee/5/tutorial/doc/bncjh.html. (Zitiert auf Seite 204)

[Ora14] Oracle. Java Naming and Directory Interface (JNDI), 2014. URL http://docs.oracle.com/javase/7/docs/technotes/guides/jndi/index.html. (Zitiert auf Seite 176)

[Pap03] M. Papazoglou. Service-oriented computing: concepts, characteristics and directions. In Proceedings of the 4th International Conference on Web Information Systems Engineering, S. 3–12. 2003. (Zitiert auf Seite 30)

[PBA+08] D. C. Plummer, T. J. Bittman, T. Austin, D. W. Cearley, D. M. Smith. Cloud Computing: Defining and Describing an Emerging Phenomenon. Gartner Research, 2008. (Zitiert auf Seite 33)

[PBMW99] L. Page, S. Brin, R. Motwani, T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford InfoLab, 1999. (Zitiert auf Seite 138)

[PDP+07] D. Pollet, S. Ducasse, L. Poyet, I. Alloui, S. Cimpan, H. Verjus. Towards A Process-Oriented Software Architecture Reconstruction Taxonomy. In Proceedings of the 11th European Conference of Software Maintenance and Reengineering, S. 137–148. 2007. (Zitiert auf Seite 51)

[Pia09] J. Piazza. Computing Migration Strategies. Whitepaper, Savvis, 2009. (Zitiert auf den Seiten 59 und 60)

[PJ11] B. Pfitzmann, N. Joukov. Migration to Multi-image Cloud Templates. In Proceedings of the IEEE International Conference on Services Computing, S. 80–87. 2011. (Zitiert auf Seite 64)

[Pup09] Puppet Labs. The Facter Program Quickly Gathers Basic Node Information, 2009. URL http://puppetlabs.com/facter. (Zitiert auf Seite 56)

[Pup14] Puppet Labs. Automate IT, 2014. URL http://puppetlabs.com/. (Zitiert auf Seite 40)

[PW03] J. Parsons, Y. Wand. Property-Based Semantic Reconciliation of Heterogeneous Information Sources. In S. Spaccapietra, S. T. March, Y. Kambayashi, Herausgeber, Conceptual Modelling (ER 2002), Band 2503 von Lecture Notes in Computer Science, S. 351–364. Springer Berlin Heidelberg, 2003. (Zitiert auf Seite 138)

[PX13] C. Pahl, H. Xiong. Migration to PaaS clouds - Migration process and architectural concerns. In Proceedings of the 7th IEEE International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, S. 86–91. 2013. (Zitiert auf den Seiten 45, 63 und 201)

[PXW13] C. Pahl, H. Xiong, R. Walshe. A Comparison of On-Premise to Cloud Migration Approaches. In K.-K. Lau, W. Lamersdorf, E. Pimentel, Herausgeber, Service-Oriented and Cloud Computing, Band 8135 von Lecture Notes in Computer Science, S. 212–226. Springer Berlin Heidelberg, 2013. (Zitiert auf den Seiten 66 und 67)

[Red14] Red Hat, Inc. GlusterFW – Write once, read everywhere, 2014. URL http://www.gluster.org/. (Zitiert auf Seite 179)

[Ree09] G. Reese. Cloud Application Architectures: Building Applications and Infrastructure in the Cloud. O'Reilly Media, Inc., 2009. (Zitiert auf den Seiten 16, 34 und 35)

[Rit12] D. Ritter. From network mining to large scale business networks. In Proceedings of the 21st international conference companion on World Wide Web, S. 989–996. 2012. (Zitiert auf Seite 50)

[RKM10] S. Ried, H. Kisker, P. Matzke. The Evolution Of Cloud Computing Markets. Technischer Bericht, Forrester Research, 2010. (Zitiert auf den Seiten 16 und 32)

[RLM+09] F. Rosenberg, P. Leitner, A. Michlmayr, P. Celikovic, S. Dustdar. Towards Composition as a Service - A Quality of Service Driven Approach. In Proceedings of the 25th IEEE International Conference on Data Engineering, S. 1733–1740. 2009. (Zitiert auf Seite 32)

[SAB+13] S. Strauch, V. Andrikopoulos, U. Breitenbücher, S. G. Sáez, O. Kopp, 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, S. 26–33. 2013. (Zitiert auf Seite 79)

[SALM12] S. Strauch, V. Andrikopoulos, F. Leymann, D. Muhler. ESBMT: Enabling Multi-Tenancy in Enterprise Service Buses. In Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science, S. 456–463. 2012. (Zitiert auf Seite 36)

[SAT+13] S. Strauch, V. Andrikopoulos, B. Thomas, D. Karastoyanova, S. Passow, K. Vukojevic-Haupt. Decision Support for the Migration of the Application Database Layer to the Cloud. In Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science, S. 639–646. 2013. (Zitiert auf Seite 79)

[Sch01] U. Schöning. Algorithmik. Spektrum, Akad. Verlag, 2001. (Zitiert auf den Seiten 104 und 107)

[Sha12] N. Shalom. Are we seeing the renaissance of enterprises in the cloud?, 2012. URL `http://highscalability.com/blog/2012/11/5/are-we-seeing-the-renaissance-of-enterprises-in-the-cloud.html`. (Zitiert auf den Seiten 16, 72 und 200)

[SJ12] L. Schubert, K. Jeffery. Advances in Clouds – Research in Future Cloud Computing. Technischer Bericht, European Union, 2012. Expert Group Report, Public verison 1.0. (Zitiert auf den Seiten 21 und 202)

[SKLU11] S. Strauch, O. Kopp, F. Leymann, T. Unger. A Taxonomy for Cloud Data Hosting Solutions. In Proceedings of the International Conference on Cloud and Green Computing, S. 577–584. 2011. (Zitiert auf Seite 79)

[SP08] K. Scheibenberger, I. Pansa. Modelling dependencies of IT Infrastructure elements. In Proceedings of the 3rd IEEE/IFIP International

Workshop on Business-driven IT Management, S. 112–113. 2008. (Zitiert auf Seite 48)

[SP11] P. Saripalli, G. Pingali. MADMAC: Multiple Attribute Decision Methodology for Adoption of Clouds. In Proceedings of the 4th IEEE International Conference on Cloud Computing, S. 316–323. 2011. (Zitiert auf den Seiten 73 und 76)

[SSS+11] M. Sethi, N. Sachindran, M. Soni, M. Gupta, P. Gupta. A framework for migrating production snapshots of composite applications to virtualized environments, S. 578–585. Institute of Electrical and Electronics Engineers, 2011. (Zitiert auf Seite 64)

[SSSL12] M. Smit, M. Shtern, B. Simmons, M. Litoiu. Partitioning Applications for Hybrid and Federated Clouds. In Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research, S. 27–41. 2012. (Zitiert auf Seite 73)

[Sug14] SugarCRM Deutschland GmbH. SugarCRM – CRM Software & Online Customer Relationship Management, 2014. URL http://sugarcrm.com/. (Zitiert auf Seite 156)

[SWH10] H. Sneed, E. Wolf, H. Heilmann. Softwaremigration in der Praxis: Übertragung alter Softwaresysteme in eine moderne Umgebung. Dpunkt.Verlag GmbH, 2010. (Zitiert auf den Seiten 16, 58, 59, 60, 68, 200 und 210)

[The06] The Open Group. TOGAF - Phase C: Information Systems Architectures - Applications Architecture, 2006. URL http://pubs.opengroup.org/architecture/togaf8-doc/arch/chap09.html. (Zitiert auf Seite 47)

[The07] The U.S. National Archives and Records Administration. The Soundex Indexing System, 2007. URL http://www.archives.gov/research/census/soundex.html. (Zitiert auf Seite 137)

[The13] The Open Group. ArchiMate 2.1, 2013. URL http://pubs.opengroup.org/architecture/archimate2-doc/. (Zitiert auf Seite 47)

[TLF+11] V. Tran, K. Lee, K. Fekete, A. Liu, J. Keung. Size Estimation of Cloud Migration Projects with Cloud Migration Point (CMP). In Proceedings of the International 5th Symposium on Empirical Software Engineering and Measurement, S. 265–274. 2011. (Zitiert auf Seite 75)

[TLMB10] I. Trummer, F. Leymann, R. Mietzner, W. Binder. Cost-Optimal Outsourcing of Applications into the Clouds. In Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science, S. 135–142. 2010. (Zitiert auf den Seiten 73 und 75)

[TUS11] B. C. Tak, B. Urgaonkar, A. Sivasubramaniam. To Move or Not to Move: The Economics of Cloud Computing. In Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing. 2011. (Zitiert auf den Seiten 73 und 75)

[TVS02] A. S. Tanenbaum, M. Van Steen. Distributed systems, Band 2. Prentice Hall, 2002. (Zitiert auf Seite 31)

[Vaa14] Vaadin Ltd. Vaadin – thinking of U and I, 2014. URL `https://vaadin.com/home`. (Zitiert auf Seite 176)

[Var10] J. Varia. Migrating Your Existing Applications to the AWS Cloud. Technischer Bericht, Amazon Web Services, 2010. (Zitiert auf Seite 60)

[Vat13] M. Vatkina. JSR 318: Enterprise JavaBeans 3.1, 2013. URL `https://jcp.org/en/jsr/detail?id=318`. (Zitiert auf Seite 176)

[VBB11] W. Voorsluys, J. Broberg, R. Buyya. Introduction to cloud computing. Cloud computing: Principles and paradigms, S. 2–44, 2011. (Zitiert auf den Seiten 34 und 35)

[VHH12] G. Vossen, T. Haselmann, T. Hoeren. Cloud-Computing für Unternehmen: Technische, wirtschaftliche, rechtliche und organisatorische Aspekte. Dpunkt.Verlag GmbH, 2012. (Zitiert auf Seite 62)

[W3C07] W3C. Web Services Policy 1.5 - Framework, 2007. URL `http://www.w3.org/TR/ws-policy/`. (Zitiert auf Seite 30)

[WAB+10] C. Ward, N. Aravamudan, K. Bhattacharya, K. Cheng, R. Filepp, R. Kearney, B. Peterson, L. Shwartz, C. Young. Workload Migration

into Clouds Challenges, Experiences, Opportunities. In Proceesings of the IEEE International Conference on Cloud Computing, S. 164–171. 2010. (Zitiert auf Seite 64)

[WBB+13] J. Wettinger, M. Behrendt, T. Binz, U. Breitenbücher, G. Breiter, F. Leymann, S. Moser, I. Schwertle, T. Spatzier. Integrating Configuration Management with Model-Driven Cloud Management Based on TOSCA. In Proceedings of the 3rd International Conference on Cloud Computing and Service Science, S. 437–446. 2013. (Zitiert auf den Seiten 26 und 42)

[WBMS10] K. Winter, S. Buckl, F. Matthes, C. Schweda. Investigating the state-of-the-art in enterprise architecture management methods in literature and practice. In Proceedings of the 5th Mediterranean Conference on Information Systems, Band 90. 2010. (Zitiert auf den Seiten 46 und 50)

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. Ferguson. Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall PTR Upper Saddle River, 2005. (Zitiert auf Seite 30)

[Wig12] A. Wiggins. The Twelve-Factor App, 2012. URL `http://12factor.net/`. (Zitiert auf Seite 85)

[Wil12] B. Wilder. Cloud Architecture Patterns. Oreilly and Associate Series. O'Reilly, 2012. (Zitiert auf den Seiten 36, 37, 63 und 76)

[Win99] W. E. Winkler. The State of Record Linkage and Current Research Problems. Technischer Bericht, Statistical Research Division, U.S. Census Bureau, 1999. (Zitiert auf Seite 138)

[WKL11] S. Wagner, O. Kopp, F. Leymann. Towards Choreography-based Process Distribution In The Cloud. In Proceedings of the IEEE International Conference on Cloud Computing and Intelligence Systems, S. 490–494. 2011. (Zitiert auf den Seiten 83 und 191)

[Wor14] WordPress Foundation. WordPress – Blog Tool, Publishing Platform, and CMS, 2014. URL `http://wordpress.org/`. (Zitiert auf Seite 156)

[WS96] R. Y. Wang, D. M. Strong. Beyond accuracy: What data quality means to data consumers. Journal of Management Information Systems, 12(4):5–33, 1996. (Zitiert auf den Seiten 119 und 120)

[YBDS08] L. Youseff, M. Butrico, D. Da Silva. Toward a Unified Ontology of Cloud Computing. In Proceedings of the Grid Computing Environments Workshop, S. 1–10. 2008. (Zitiert auf Seite 31)

Alle Onlineverweise wurden zuletzt am 20.01.2015 aufgerufen.

Abbildungsverzeichnis

1.1 Übersicht der Forschungsbeiträge 17

2.1 Relation der verschiedenen Mengen von Anwendungen 37
2.2 Beispiel TOSCA TopologyTemplate 43
2.3 Klassifikation von Ansätzen zur Anwendungsmigration 59

3.1 Klassifikation der AROMA-Methode 68
3.2 AROMA-Methode 70
3.3 AROMA-Methode Variante 1 81
3.4 AROMA-Methode Variante 2 82

4.1 Beispiel-ETG der Anwendung „Moodle“ 90
4.2 Enterprise Topologie Graph Metamodell 91
4.3 Darstellung eines exemplarischen Topologie Queries 103
4.4 Beispiel für Reduktion durch Anpassung der Relation 110
4.5 DBMS als gemeinsam genutzte Komponente 111
4.6 Lösungen für gemeinsam genutzte Komponenten 113

5.1 Wachsender ETG während eines Crawler-Laufes 123
5.2 Architektur einer Crawler-Umgebung 125

5.3 ETG-Repräsentation eines BPEL-Geschäftsprozesses 142
5.4 ETG-Repräsentation Webanwendung und Webservers 143
5.5 ETG-Repräsentation einer Beispiel-ESB-Route 148
5.6 ETG-Repräsentation eines Betriebssystems und einer VM . . . 150
5.7 Entwicklungs- und Testmethode 152

6.1 Übersicht des OpenTOSCA-Ökosystems 165

7.1 Gesamtarchitektur 176
7.2 ETG-Framework 177
7.3 Architektur der ETG-Verwaltung 178
7.4 Benutzeroberfläche ETG-Verwaltung 180
7.5 Architektur des ETG-Crawlers 181
7.6 Benutzeroberfläche Crawler 183
7.7 Architektur des Migrationsassistenten 187
7.8 Migrationslösung im AROMA-Migrationsassistenten 190
7.9 Geschäftsprozess im AROMA Migrationsassistenten 192

8.1 Enterprise Topologie Graph der Moodle enthält 194
8.2 Zuweisung der gewählten Anbieter zu den Komponenten . . . 196
8.3 Verschiedene Lösungstopologien 197
8.4 Übersicht des Ansatzes von Breitenbücher et al. 198

9.1 Übersicht der Forschungsbeiträge 208

Definitionsverzeichnis

2.1 Anwendung – informell 31
2.2 Cloud-Anwendung – informell 37
2.3 Native Cloud-Anwendung – informell 37
2.4 Anwendungstopologie – informell 38
2.5 Enterprise Topologie – informell 46
2.6 Anwendungsmigration – informell 58

4.1 Enterprise Topologie Graph 92
4.2 Komponenten 94
4.3 Relationen 94
4.4 Relationen von/nach 94
4.5 Ein-/ausgehende Relationen 95
4.6 Komponententypen 95
4.7 Relationentypen 95
4.8 Vererbung von Komponententypen 95
4.9 Vererbung von Relationentypen 96
4.10 Zuweisung von Komponententypen 96
4.11 Zuweisung von Relationentypen 96
4.12 Alle Kindtypen eines Komponententyps 97
4.13 Alle Kindtypen eines Relationentyps 97

4.14 Elemente 98
4.15 Eigenschaften 98
4.16 Grundlegende Relationentypen 100
4.17 Segment 100
4.18 Segmente eines ETGs 101
4.19 Ein-/ausgehende Relationen in Segmenten 101

5.1 Zum Komponententyp kompatible Plugins 128
5.2 Abhängigkeit von Crawler-Plugins auf den ETG 129
5.3 Crawler-Iteration 129
5.4 Versionszähler 129
5.5 Letzte Ausführung von Plugin auf Komponente 130

Algorithmenverzeichnis

4.1 *topologieQuery*($k \in K, tiefe \in \mathbb{N}_0$) 105
4.2 *pruefeEigenschaften*($k \in K$) 106
4.3 *deepDive*($K_{eingabe} \subseteq K$) 108
4.4 *gemeinsamGenutzteKomponenten*($K_a \subseteq K$) 112

5.1 *crawlETG*($etg \in \mathcal{ETG}$) 130
5.2 *crawlerIteration*($etg \in \mathcal{ETG}, it \in \mathbb{N}_0$) 131
5.3 *naechstesPlugin*($k \in K, it \in \mathbb{N}_0$) 132
5.4 *crawlETGSelektiv*($etg \in \mathcal{ETG}$) 134
5.5 *etgAktualisieren*($etg \in \mathcal{ETG}$) 136

Liste mathematischer Symbole und Funktionen

Symbol oder Funktion	Bedeutung
ausgehendeRelationen	Funktion, die alle ausgehenden Relationen einer Komponente zurückgibt (Definition 4.5, Seite 95)
crawlETG	Funktion, die einen Crawler-Lauf durchführt (Algorithmus 5.1, Seite 130)
crawlETGSelektiv	Funktion, die einen selektiven Crawler-Lauf durchführt, d.h. auf einen Teil der IT beschränkt ist (Algorithmus 5.4, Seite 134)
crawlerIteration	Funktion, die eine Iteration eines Crawler-Laufes durchführt (Algorithmus 5.2, Seite 131)
deepDive	Funktion, die alle durch ausgehende Relationen erreichbare Komponenten bestimmt (Algorithmus 4.3, Seite 108)
Elemente	Menge der Elemente eines ETGs, welche sich aus der Menge aller Komponenten und Relationen zusammensetzt (Definition 4.14, Seite 98)

Symbol oder Funktion	Bedeutung
etg	Enterprise Topologie Graph (Definition 4.1, Seite 92)
etg_t	Enterprise Topologie Graph, der zum Zeitpunkt t erstellt wurde (Definition 4.1, Seite 92)
$\mathcal{ETG}$	Menge aller ETGs (Definition 4.1, Seite 92)
etgAbhaengigkeiten	Funktion welche die Abhängigkeiten eines Crawler-Plugins auf den ETG zurückgibt, repräsentiert durch eine Menge von Topologie Queries (Definition 5.2, Seite 129)
eingehendeRelationen	Funktion, die alle eingehenden Relationen einer Komponente zurückgibt (Definition 4.5, Seite 95)
erbtVonKT	Funktion, die für einen Komponententypen zurückgibt, von welchem dieser erbt (Definition 4.8, Seite 95)
erbtVonRT	Funktion, die für einen Relationentyp zurückgibt, von welchem dieser erbt (Definition 4.9, Seite 96)
erbenVonKT	Funktion, die alle Komponententypen zurückgibt, die von einem gegebenen Komponententypen erben (Definition 4.12, Seite 97)
erbenVonRT	Funktion, die alle Relationentypen zurückgibt, die von einem gegebenen Relationentypen erben (Definition 4.13, Seite 97)
hatEigenschaften	Funktion, welche die Eigenschaften einer Komponente oder Relation zurückgibt (Definition 4.15, Seite 98)

Symbol oder Funktion	Bedeutung
it	Iteration eines Crawler-Laufes (Definition 5.3, Seite 129)
k bzw. k_i	Komponente (Definition 4.2, Seite 94)
K	Menge der Komponenten eines ETGs (Definition 4.2, Seite 94)
K_i	Teilmenge der Komponenten eines ETGs
KT	Menge der Komponententypen eines ETGs (Definition 4.6, Seite 95)
kompatiblePlugins	Funktion, die alle zu einem Komponententyp kompatiblen Crawler-Plugins zurückgibt (Definition 5.1, Seite 128)
la	Funktion, die zurückgibt, in welcher Iteration ein Crawler-Plugin das letzte Mal auf einer Komponente ausgeführt wurde (Definition 5.5, Seite 130)
N	Menge aller möglichen Namen einer Eigenschaft (Definition 4.15, Seite 98)
naechstesPlugin	Funktion, die für eine Komponente das Crawler-Plugin zurückgibt, welches als nächstes ausgeführt werden soll (Algorithmus 5.3, Seite 132)
nach	Funktion, die für eine Relation zurückgibt, von welcher Komponente diese ausgeht (Definition 4.4, Seite 94)
plugin	Crawler-Plugin
P	Menge aller Crawler-Plugins (Definition 5.1, Seite 128)

Symbol oder Funktion	Bedeutung
r bzw. r_i	Relation (Definition 4.3, Seite 94)
R	Menge der Relationen eines ETGs (Definition 4.3, Seite 94)
R_i	Teilmenge der Relationen eines ETGs
RT	Menge der Relationentypen eines ETGs (Definition 4.7, Seite 95)
s bzw. s_j	Segment eines ETGs (Definition 4.18, Seite 101)
S	Menge der Segmente eines ETGs (Definition 4.18, Seite 101)
typK	Funktion, die einer Komponente einen Komponententypen zuweist (Definition 4.10, Seite 96)
typR	Funktion, die einer Relation einen Relationentypen zuweist (Definition 4.11, Seite 96)
vz_{it}	Funktion, die für jede Komponente, Relation oder Eigenschaft deren Versionszähler nach der *it*-ten Iteration des Crawlers zurückgibt (Definition 5.4, Seite 129)
von	Funktion, die für eine Relation zurückgibt, in welcher Komponente diese eingeht (Definition 4.4, Seite 94)
W	Menge aller möglichen Werte einer Eigenschaft (Definition 4.15, Seite 98)
Zeichen	Menge aller zulässigen Zeichen (Definition 4.15, Seite 98)